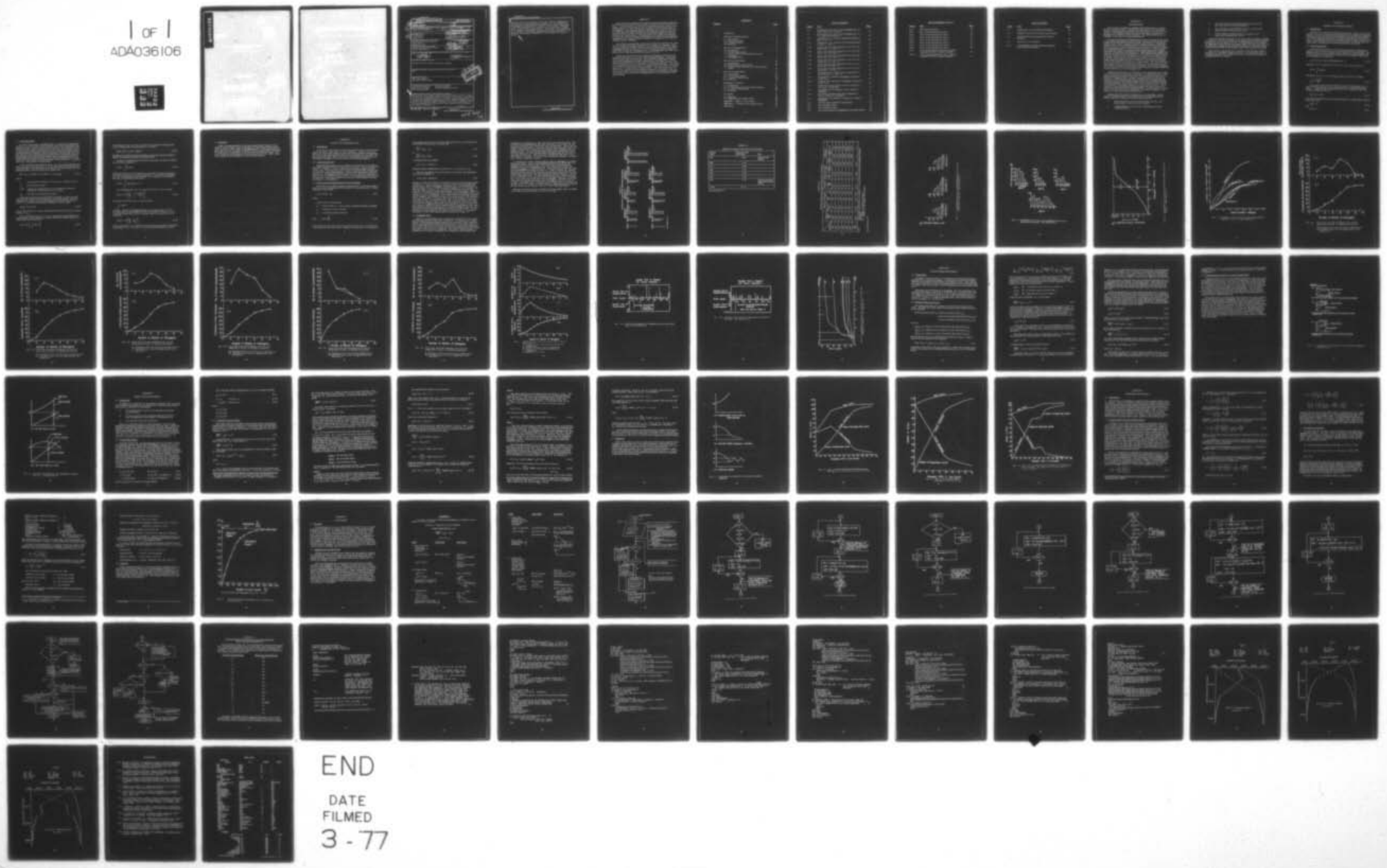
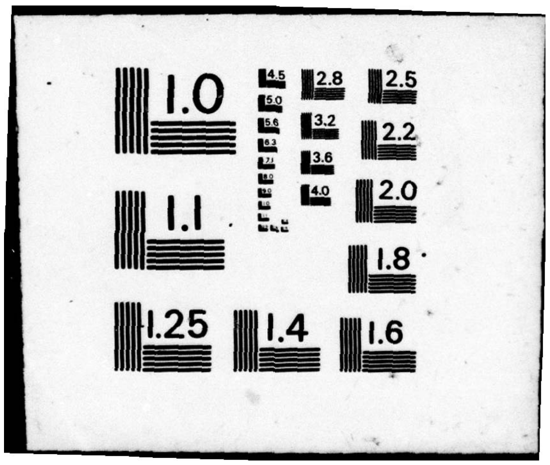
AD-A036 106   POLYTECHNIC INST OF NEW YORK BROOKLYN DEPT OF ELECTR--ETC F/G 9/2
EFFECT OF MANPOWER DEPLOYMENT AND BUG GENERATION ON SOFTWARE ER--ETC(U)
JAN 77 M L SHOOMAN, S NATARAJAN   F30602-74-C-0294

UNCLASSIFIED   POLY-EE/EP-76-007   RADC-TR-76-400   NL

1 OF 1
ADA036106



END
DATE
FILMED
3-77


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

AD A036106

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
18 1. REPORT NUMBER RADC-TR-76-400 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 9
6 4. TITLE (and Subtitle) EFFECT OF MANPOWER DEPLOYMENT AND BUG GENERATION ON SOFTWARE ERROR MODELS.	5. TYPE OF REPORT & PERIOD COVERED Interim Report. 1 Jul 75 — 30 Jun 76
	14 6. PERFORMING ORG. REPORT NUMBER Poly-EE/EP-76-007
10 7. AUTHOR(s) M. L./Shooman S./Natarajan	15 8. CONTRACT OR GRANT NUMBER(s) F30602-74-C-0294
9. PERFORMING ORGANIZATION NAME AND ADDRESS Polytechnic Institute of New York 333 Jay Street Brooklyn NY 11201	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 16 62702F 55500806 17 08
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441	11 12. REPORT DATE Jan[redacted] 1977
	13. NUMBER OF PAGES 77
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same 12 85p.	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Same

DDC
RECEIVED
FEB 28 1977
C

18. SUPPLEMENTARY NOTES
RADC Project Engineer:
Capt Alan N. Sukert (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Software Error Modeling
Error Correction Model
Error Generation Model
Software Reliability
Software Debugging Cost Model

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
Early software error models by Shooman and Jelinski-Moranda related the number of errors in a large software system to the rate of error removal. Expressions for the number of remaining errors as the software undergoes debugging were formulated and additional assumptions were made to relate the number of residual errors to the operational system reliability. One of the key assumptions of the above models was that the sum of the errors removed and those remaining in the program is constant.

(cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

over

408 717 LB

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

This report adds a major refinement to the above models by introducing the possibility of error generation during debugging. In this refinement the error generation terms are modeled in several different ways: proportional to the number of detected errors, corrected errors, the number of remaining errors, or some function of these effects. The correction rate is assumed to be a function of the manpower deployed on the project, thus permitting the use of the model to investigate optimum manpower deployment strategies. The effects on the economics of debugging due to error growth have also been analyzed.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

ABSTRACT

Several previous models in the literature have discussed how the number of errors in a large software system is related to the rate of error removal. In 1971 Shooman, and Jelinski and Moranda proposed similar Probabilistic Models for the removal rate of software errors during software development. The models proposed by Shooman were based on error data on 7 different large operating systems and application programs and collected by Hesse, and these models also fit the data of Akiyama which was collected on small programs. Expressions for the number of remaining errors as the software undergoes debugging were formulated and additional assumptions were made to relate the number of residual errors to the operational system reliability.

One of the key assumptions in the above models was that the sum of the errors removed and those remaining in the program is a constant. Thus, if we can estimate the initial number of errors in the system at the start of debugging and keep careful records of those removed we have a good estimate of the number of remaining errors. In 1973 Shooman described a test procedure for estimating the initial number of errors.

In this work we add a major refinement to the above models by introducing the possibility of error generation during debugging. A generated error is due to one of two causes: (1) a bug whose correction is invalid and further debugging on the same statements is essential, (2) a new bug which is generated as the result of the correction of a different error. The error generation terms are modeled in several different ways: proportional to the number of detected errors, corrected errors, the number of remaining errors, or some function of these effects. The correction rate is assumed to be a function of the manpower deployed on the project, thus, one can use the model to investigate optimum manpower deployment strategies. The effects on the economics of debugging due to error growth have also been analyzed.

CONTENTS

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

CHAPTER 1
INTRODUCTION

The concept of reliability as applied to the performance of computer programs is in its infancy. Software packages with an ever increasing degree of complexity have emerged in the last decade. It is therefore imperative to have highly reliable software systems when large software packages are required to perform complex, real-time operations as, for example, in the case of a moonlanding mission.

The theory of Software Reliability differs from that of Hardware Reliability in that system failure is not due to part failures (which are due to a variety of causes) but is caused by software bugs which are really latent design errors. Once a software defect is properly fixed it is in general fixed for all time. Failure usually occurs only when a program is exposed to an environment that it was not designed or tested for. The large number of possible states of a program and its inputs make perfect comprehension of the program requirements and implementation and complete testing of the program generally impossible. Thus Software Reliability is essentially a measure of the confidence we have in the design and it's ability to function properly in all environments it is expected to be subjected to. In the life cycle of software there are generally one or more test phases during which reliability improves as errors are identified and corrected. The first few months of operation may include either: (1) a non-growth operational phase during which further corrections are not made (for practical and economic reasons) and reliability is constant, or (2) further debugging and correction of errors as they occur which essentially amounts to the use of early operation as a final debugging test.

Reliability predictions are made based on the estimation of the number of errors present in a program. In Chapter 2 is discussed some of the pioneering work [1-3] done in this field. The approach used there was to study the debugging history of previous programs similar to the one in question. From these one determined constants for the model and made predictions in the initial planning stages of the system reliability and the required debugging. Any change to increase the accuracy of the estimate was made after error data was available for the current project. The error models were developed with the assumption that no new errors are added to a program during it's debugging phase. In other words the cumulative error correction curve approaches a horizontal asymptote. This however is contrary to practical experience.

Discussions with program managers have revealed that a certain amount of bug generation is associated with every debugging process. Some of the ways in which errors may be generated are

1. The correction of a bug may work locally only (i. e., the global aspects of the error still remain).

2. A typographical error may arise invalidating the result of bug correction.

3. The correction is based upon faulty analysis, thus complete bug removal is not accomplished.

4. The correction is accomplished, however, it is accompanied by the creation of a new error.

5. Errors which are detected but not corrected, act in many ways like generated errors.

During the development phase we are faced with two classes of changes in a program, those due to changes in the specifications (design) and those necessary to correct software errors. Although both classes of changes are important we only discuss the changes needed for error correction. In evaluating the constants of the models from data, it is important to keep careful records so one is able to differentiate between these two effects.

The concept of bug generation is analyzed in the following chapters. In Chapter 5, a two-phase model for the error correction rate leads to various possibilities for a hypothetical model. An economic analysis of debugging follows in Chapter 6.

CHAPTER 2

SUMMARY OF EXISTING MODELS

2.1 Introduction

Earlier error models developed in the literature assume that the total number of errors in a program is fixed and that if we record the cumulative number of errors corrected during debugging, then the difference between the initial number and the number corrected represents the remaining errors. The reliability models to be discussed in this chapter relate the probability of encountering a software bug to the number of residual bugs, the total number of instructions and the instruction processing rate.

2.2 Error Decay Model

Assuming that a careful record of errors corrected is maintained, a graph of error correction rate versus debugging time may be drawn. The debugging process starts with the number of errors corrected being equal to zero, and the time axis starts at the instant the debuggers begin work. The debugging rate is defined as

$$r_c(\tau) = \text{errors removed/debugging time } \tau \tag{2.1}$$

Using Eq. (2.1) a cumulative error curve, $n_c(\tau)$, can be defined as

$$n_c(\tau) = \int_0^\tau r_c(x)\,dx \tag{2.2}$$

Solving Eq. (2.2) for $r_c(\tau)$, the slope of the $n_c(\tau)$ curve, we obtain

$$r_c(\tau) = \frac{dn_c(\tau)}{d\tau} \tag{2.3}$$

If we assume that the total number of errors in the program, n_t, remains constant, then the curve $n_c(\tau)$ approaches n_t asymptotically for large τ. Assuming that all detected errors are corrected we can write for the remaining errors $n(\tau)$

$$n(\tau) = n_t - n_c(\tau) \tag{2.4}$$

If we further assume that in any sizable program it is impossible to remove all errors, then

$$n_c(\tau) < n_t \tag{2.5}$$

and

$$n(\tau) > 0 \tag{2.6}$$

2.3 Reliability Model

In order to formulate a reliability model, we assume that operational software errors occur due to the occasional traversing of a portion of the program in which a hidden software bug is present. An expression for the probability that a bug is encountered in the time interval Δt after t hours of successful operation may be derived. This must be proportional to the probability that any randomly chosen instruction contains a bug. If we further assume that all instructions are equally likely, and that all bugs are homogeneously and randomly distributed, then the probability that a randomly chosen instruction contains a bug is given by the number of errors normalized with respect to the total number of instructions $e_c(\tau) = n_c(\tau)/I_T$ where I_T = total number of instructions.

From reliability and probability theory [11] it is obvious that the probability of failure in the interval t to $t + \Delta t$, given that no failures have occurred till time t is proportional to the failure rate (hazard function) $Z(t)$. Mathematically we may write the above argument as

$$P[t < t_f \leq (t + \Delta t)/t_f > t] = Z(t)\,\Delta t = k_1\, e_r(\tau)\, r_p \Delta t \tag{2.7}$$

where

t_f	= operating time to failure (occurrence of a software error)
k_1	= an arbitrary constant
$e_r(\tau)$	= number of remaining errors normalized with respect to the number of instructions, i.e., $n(\tau)/I_T$
r_p	= the rate of instruction processing

In the past there have been difficulties in defining r_p since for each loop, the rate at which the instructions are processed varies. Also if interruptions or jumps or calls to subroutines occur the processing rate varies. To overcome this problem a simplified model is used,

$$z(t)\,\Delta t = k\, e_r(\tau)\, \Delta t \tag{2.7a}$$

where $k \triangleq$ an arbitrary constant which must be measured for the particular program $= k_1 r_p$.

From reliability theory [11] it can be shown that the probability of no system failures in the interval (o, t) is the reliability function which is related to the hazard function by

$$R(t) = \exp\left\{ - \int_0^t z(x)\, dx \right\} \tag{2.8}$$

Substituting for z(t) from Eq. (2.7a) into Eq. (2.8) and assuming k and $e_r(\tau)$ are independent of operating time t we obtain

$$R(t) = \exp\{-k\, e_r(\tau) t\} = \exp(\gamma t) \tag{2.9}$$

Equation (2.9) states that the probability of successful operation without bugs is an exponential function of operating time.

Using the reliability model the MTTF, mean time to (software) failure can be easily computed [1]

$$\text{MTTF} = \int_0^{\infty} R(t)\, dt \tag{2.10}$$

Explaining the above set of equations we find that to measure the parameters in the model we have to work backwards. By operating the program in a real or simulated manner MTTF [1] can be observed. Substitution from Eq. (2.9) into Eq. (2.10) yields

$$\text{MTTF} = \int_0^{\infty} \exp(-\gamma t)\, dt = 1/\gamma \tag{2.11}$$

By substituting Eqs. (2.2, 2.4 and 2.9) into Eq. (2.11) we obtain

$$\text{MTTF} = 1/k \left[\frac{n_t}{I_T} - \int_0^{\tau} \frac{r_c(x)\, dx}{I_T} \right] \tag{2.12}$$

Assuming that careful error records are kept,

$$\int_0^{\tau} r_c(x)\, dx$$

is known. The two remaining parameters to be determined, k and n_t, are determined by measuring the MTTF at two different values of t. If we further assume that $r_c(\tau)$ is a constant equal to r_o, then Eq. (2.12) reduces to

$$\text{MTTF} = 1/k \left(\frac{n_t}{I_T} - \frac{r_o}{I_T} \tau \right) \tag{2.13}$$

As we said earlier, if we work backwards with a record of MTTF and with a knowledge of r_o, τ, I_T and k, n_t can be computed using Eq. (2.13).

2.4 Summary

In the foregoing discussion it has been assumed that the total errors which is a sum of those errors remaining in the program and those eliminated remains constant during the debugging process. This assumption has been made in order to predict a behavior with less mathematical complexity. In reality there is a generation of errors associated with correction. As a result of this the total number of errors increases with time. Such a behavior of bugs will be investigated in the following chapters.

CHAPTER 3

SURVEY OF DEBUGGING DATA

3.1 Introduction

To derive accurate values for model parameters and to verify the validity of the models a wide range of data is necessary. However, one can establish the model with limited data and refine the parameters (and perhaps the form) of the model with data which will be available at a later date. The data discussed in this chapter can be used to determine the parameters of the models in Chapter 2 as well as those in the following chapters.

3.2 Large program data

Seven different programs were studied in the literature [1] and the error content (number of errors removed) was recorded. These are shown in Table 3.1. Figures 3.1 and 3.2 show the normalized changes recorded every month. From those figures we observe a generally decreasing trend in error rate versus debugging time. One possible explanation of this phenomenon is, if debugging is efficient then the errors decrease with time. If fewer errors are present then the number of errors discovered and removed are also few. This above argument is at present unsubstantiated and may be thought of as a hypothesis.

3.3 Errors removed proportional to errors remaining

One of the error models proposed by Shooman [1] is that the number of errors removed is proportional to the number of remaining errors. Mathematically we may write*

$$r_c(\tau) = k_c(n_t - n_c) \tag{3.1}$$

where

$r_c(\tau)$ = error correction rate

n_t = total errors, i.e., sum of those removed and those remaining

n_c = number of errors removed

k_c = a constant of proportionality

since $$r_c(\tau) = \frac{dn_c}{d\tau} \tag{3.2}$$

*The notation in this report differs slightly from that used in the literature.

If we assume no new errors are generated and that all errors detected are immediately and perfectly corrected, then

$$\frac{dn_c}{d\tau} = k_c(n_t - n_c) \tag{3.3}$$

or

$$\frac{dn_c}{d\tau} + k_c n_c = k_c n_t \tag{3.4}$$

A solution of Eq. (3.4) yields

$$n_c(\tau) = n_t + A \exp(-k_c \tau) \tag{3.5}$$

Using the initial condition that $n_c(\tau=0) = 0$ at $\tau = 0$ we get $A = -n_t$.

Thus the cumulative error correction curve becomes the exponential rise to the asymptote n_t

$$n_c(\tau) = n_t\{1 - \exp(-k_c \tau)\} \tag{3.6}$$

Figures 3.3 and 3.4 are plots of cumulative errors corrected from the data in Table 3.1. These curves have a profile similar to an exponential only during the later part of debugging. Therefore, Eq. (3.6) cannot entirely describe these curves. To understand the dynamics of error behavior, the data from Figs. 3.1 and 3.2 is redrawn as continuous curves in Figs. 3.5(a-f). The scales are suitably normalized. Let us postulate a model in which error correction rate per man-month decays exponentially (Fig. 3.6a) during the course of debugging. If we further assume a triangular distribution of manpower as illustrated in Fig. 3.6b, then a product of the ordinates of the curves in Figs. 3.6(a and b) gives rise to the number of errors corrected per month. The resulting curve Fig. 3.6c rate appears to be somewhat similar to those in Figs. 3.5(a-f), and an integral of this curve (Fig. 3.6d) over the debugging period gives the cumulative errors corrected. We are, however, unable to compare quantitatively the hypothetical curve with those obtained using the data, because a comparison depends upon the scales used and the scale for the hypothetical curve is not based on data. In conclusion we may say that the error model as shown in Fig. 3.6 resembles the profile of some of the experimental curves shown in Fig. 3.5.

3.4 Debugging effort

Before we can postulate an error model it is advisable to refer to other developments that have taken place in this field. An experiment on debugging of a 4000 machine language program was conducted at Bell Laboratories [6]. In conducting this experiment programmers were asked to fill a certain Trouble Report/Correction Report (TR/CR) form and another supplementary form in order to derive relevant information regarding the nature of bugs. Sixty-three such forms were completed. Out of the 63 TRs generated, 11 contained no errors and 7 contained minor annoyances which would in all probability not be defined as a bug. Thus there were 45 real errors. This works out to be approximately 1% of the total lines of code which the programmers declared to be in agreement with previous data. One of the outcomes of this experiment is a record of the time taken to remove each error. Figure 3.7 shows the working time and computer time expended in the debugging process. A careful examination of this figure reveals that except for a few spikes, the time expended while working on most of the errors is approximately the same. This leads us to conclude that the rate of error correction is a constant. We will use this assumption in Chapter 4 and prove that it leads to anomalous results.

F. Akiyama [8] reports another study on software debugging. At Fujitsu Limited in Tokyo a system called SAMPLE (running under the Monitor V of FACOM 230-60 the Fujitsu large-scale computer) was developed. The SAMPLE consists of seven modules and was programmed in FASP, the assembler language for FACOM 230-60. The program sizes and the organization of SAMPLE are shown in Table 3.2. Table 3.3 shows the relationship between the nature of the program and the occurrence of bugs for each module, while Fig. 3.8 portrays the cumulative number of bugs as a function of the development time. A careful examination of Fig. 3.5 shows that the error correction rate was not a constant all through the process of debugging. It decreased toward the later part of debugging. All these curves have a horizontal asymptote which shows that the number of errors corrected reaches a saturation level. Calculation of the cumulative error curves from the data in Ref. [1] displays a similar behavior. (Also illustrated in Fig.3.4) We will concentrate on this idea and use it in the following chapters.

Having reviewed some of the available literature in this area, we should now be able to proceed further with our error model.

TABLE 3.1

CHANGE DATA FOR SEVEN DIFFERENT PROGRAMS

	Application A 240, 000 Inst.		Application B 240, 000 Inst.		Application C 240, 000 Inst.		Application D 240, 000 Inst.	
Month	**Changes**	**Changes/Inst.**	**Changes**	**Changes/Inst.**	**Changes**	**Changes/Inst.**	**Changes**	**Changes/Inst.**
1	514	2.15×10^{-3}	905	3.76×10^{-3}	235	0.98×10^{-3}	331	1.38×10^{-3}
2	926	3.85	376	1.57	398	1.66	397	1.66
3	754	3.15	362	1.51	297	1.24	269	1.12
4	662	2.76	192	0.80	506	2.11	296	1.24
5	308	1.28	70	0.29	174	0.72	314	1.31
6	108	0.45			55	0.23	183	0.76
					60	0.25	158	0.66
							368	1.54
							337	1.41
							249	1.02
							166	0.69
							108	0.45
							31	0.13

	Supervisory A 210, 000 Inst.		Supervisory B 240, 000 Inst.		Supervisory C 230, 000 Inst.	
Month	**Changes**	**Changes/Inst.**	**Changes**	**Changes/Inst.**	**Changes**	**Changes/Inst.**
1	110	0.52×10^{-3}	250	1.04×10^{-3}	225	0.98×10^{-3}
2	238	1.14	520	2.16	287	1.24
3	185	0.88	430	1.80	497	2.16
4	425	2.02	300	1.25	400	1.7
5	325	1.55	170	0.71	180	0.78
6	37	0.18	120	0.50	50	0.22
7	5	0.02	60	0.25		
8			40	0.17		

(see Reference 1)

TABLE 3.2

SAMPLE's structure and scale of each module

Module name	Program steps (Kilo step)	Note
MA	4.03	experimental module
MB	1.32	
MC	5.45	
MD	1.67	
ME	2.05	
MF	2.51	
MT	2.10	Common program table and message string
Total	19.13	

(see Reference 8)

TABLE 3·3

Relation between the bugs and the nature of program

Module	Program Steps (S)	Nature of Program			Number of Bugs (B)	Ratio for bugs		Difficulty	
		Decision (D)	Call (J)	C (D+J)		$(\frac{B}{S}) \times 100$	$(\frac{B}{C}) \times 10$	$(\frac{D}{S}) \times 10$	$(\frac{C}{S}) \times 10$
MA	4032	372	283	655	102	2.52	1.55	0.92	1.62
MB	1329	215	44	259	18	1.35	0.69	1.61	1.94
MC	5453	552	362	914	93	1.70	1.01	1.01	1.67
MD	1674	111	130	241	26	1.55	1.07	0.66	1.43
ME	2051	315	197	512	71	3.46	1.38	1.53	2.49
MF	2513	217	186	403	37	1.47	0.91	0.86	1.60
sub total	17052	1782	1202	2984	347	2.84	1.10	----	----

note} This table excludes data of module MT (communication table and message string of SAMPLE)
D: number of decision symbols
J: number of sub-routine call symbols

(see Reference 8)

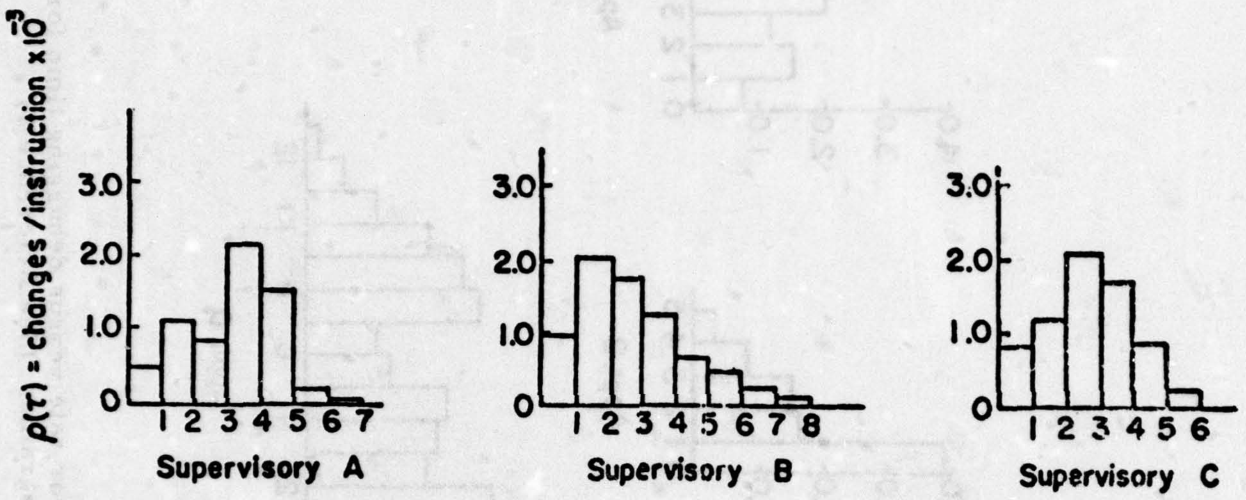


Fig. 3.1 Normalized error rate versus debugging time for three supervisory programs. (See Reference 1)

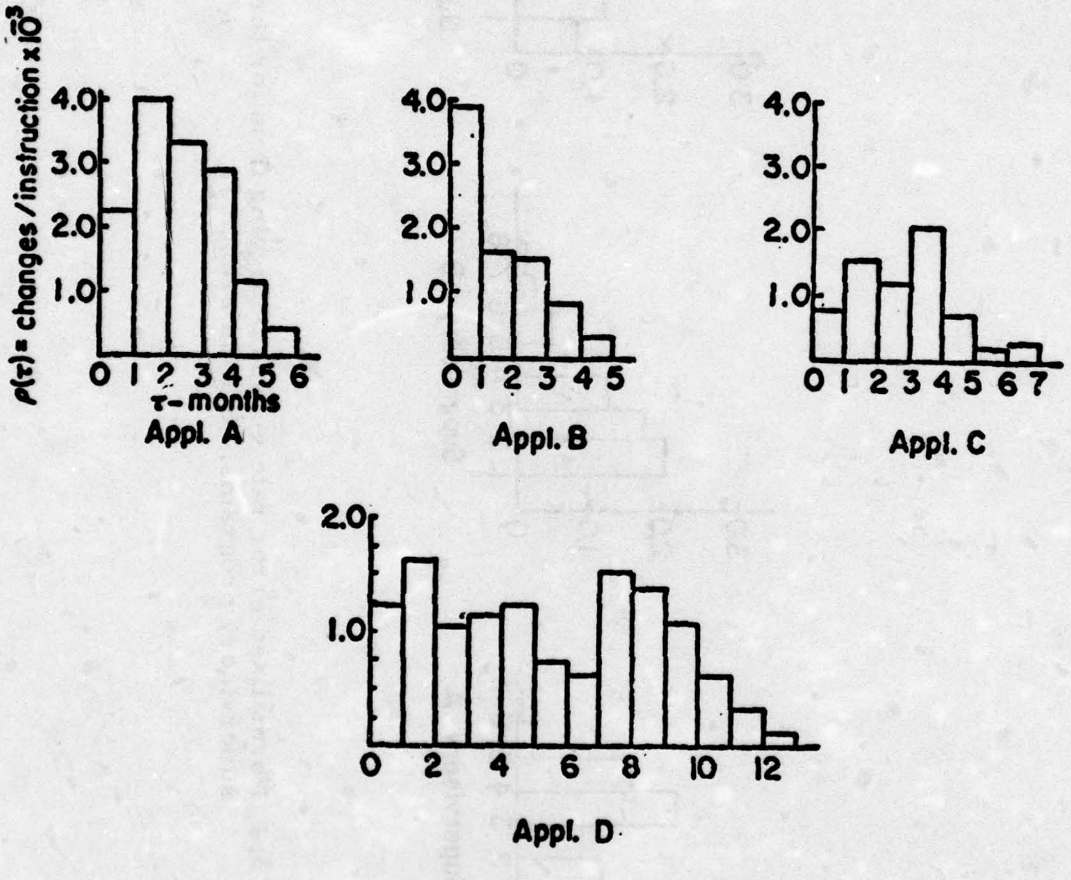


Fig. 3.2 Normalized error rate versus debugging time for four applications programs. (See Reference 1)

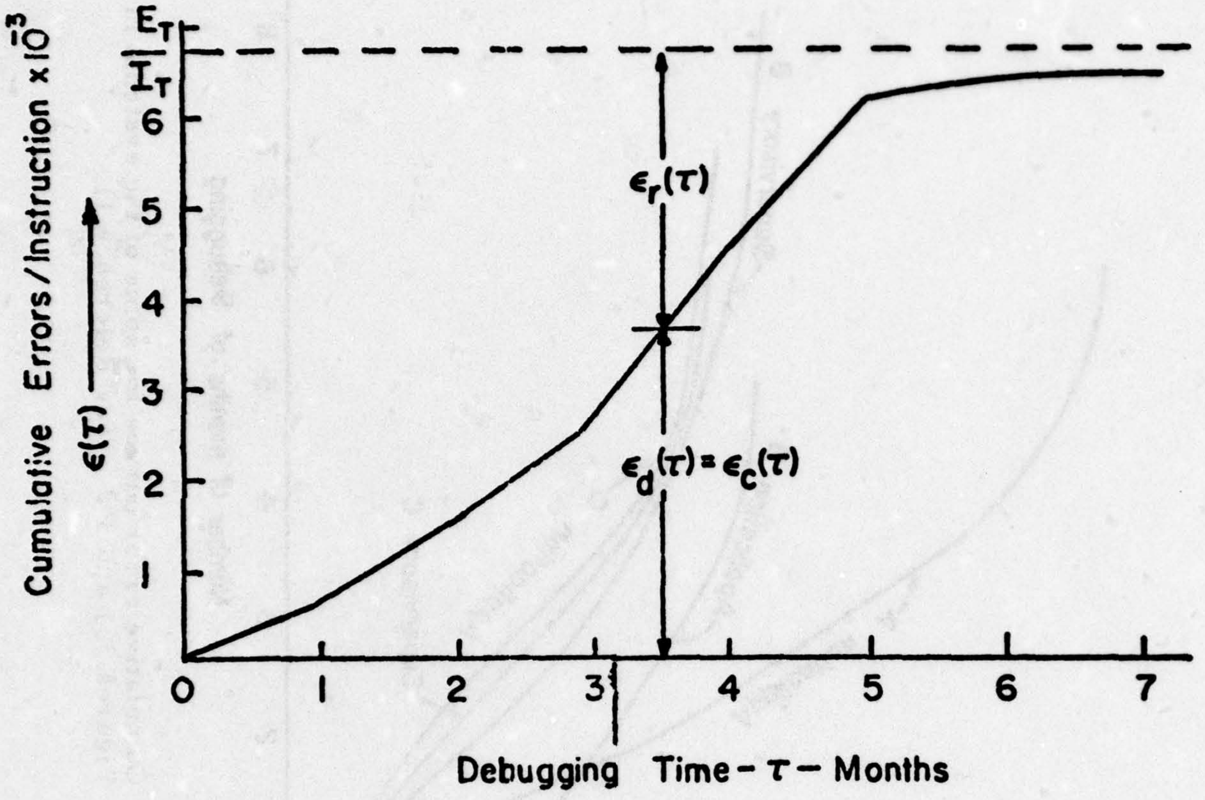


Fig. 3.3 Cumulative error curve for supervisory system A given in Figure 1. (See Reference 1)

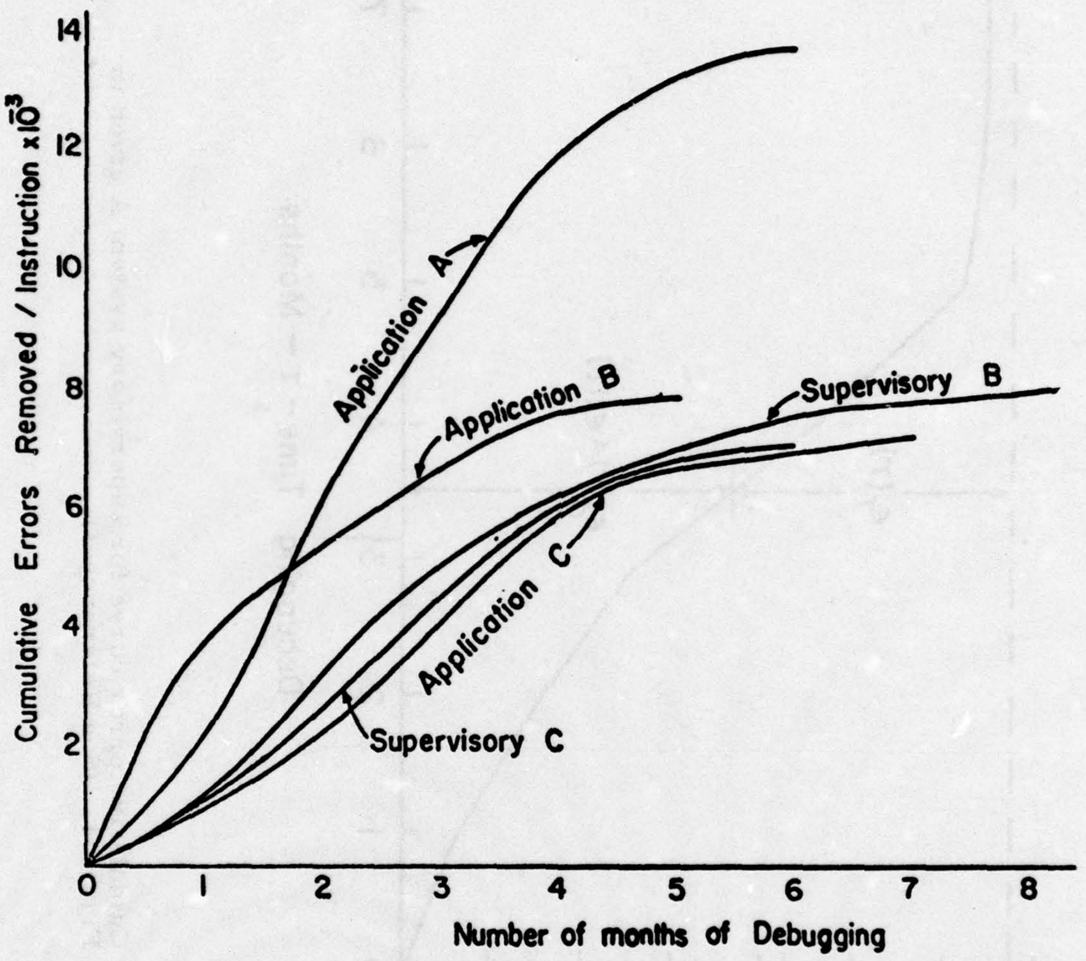


Fig. 3.4 Cumulative error curves for some of the systems in Figures 3.1 and 3.2 (See Reference 1)

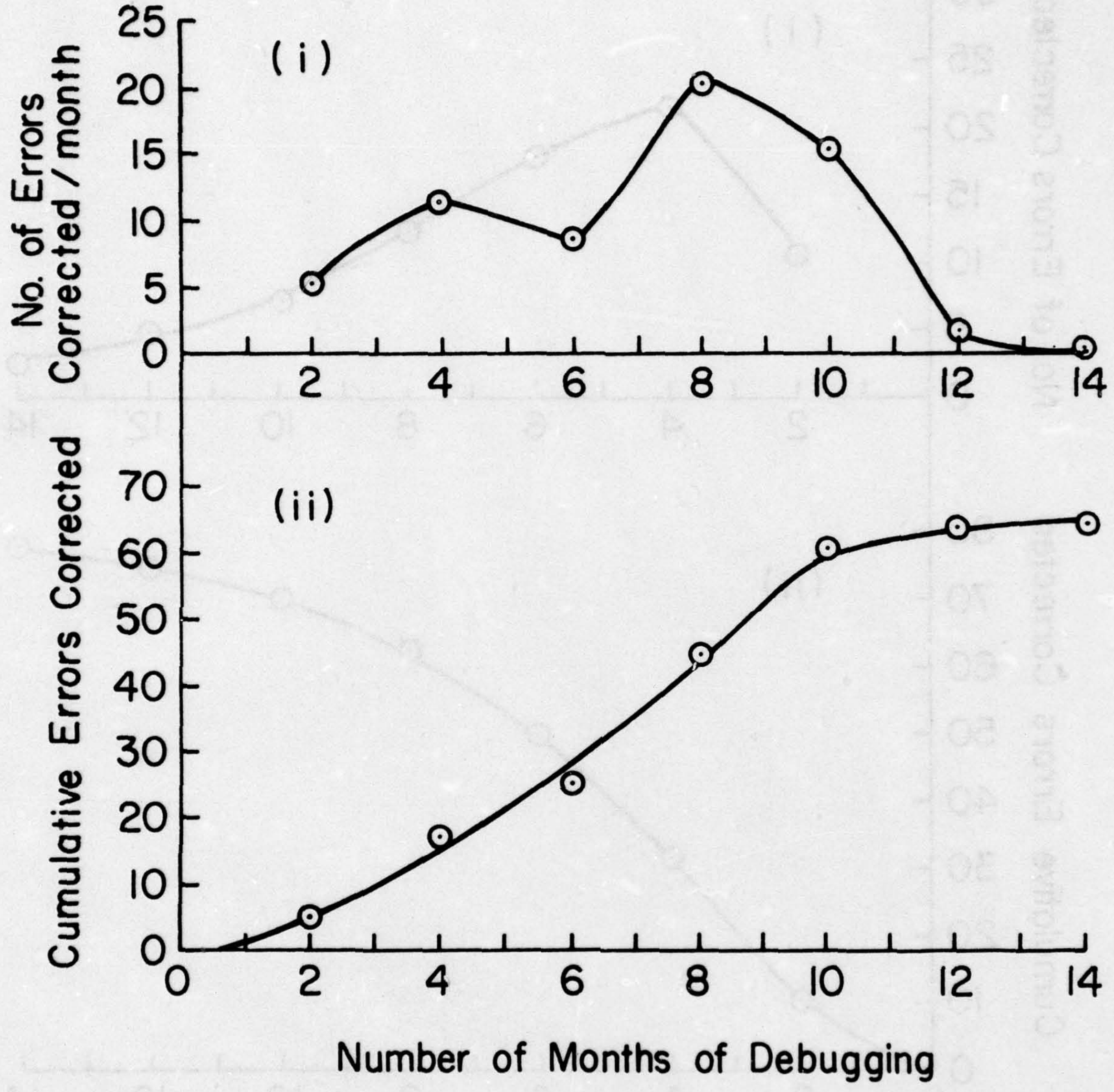


Fig. 3.5a Error rate curve and cumulative error curve for supervisory A of Fig. 3.1 (Suitably normalized)

(i) Envelope of the error rate curve for Supervisory A
(ii) Cumulative error curve for Supervisory A (see Reference 1)

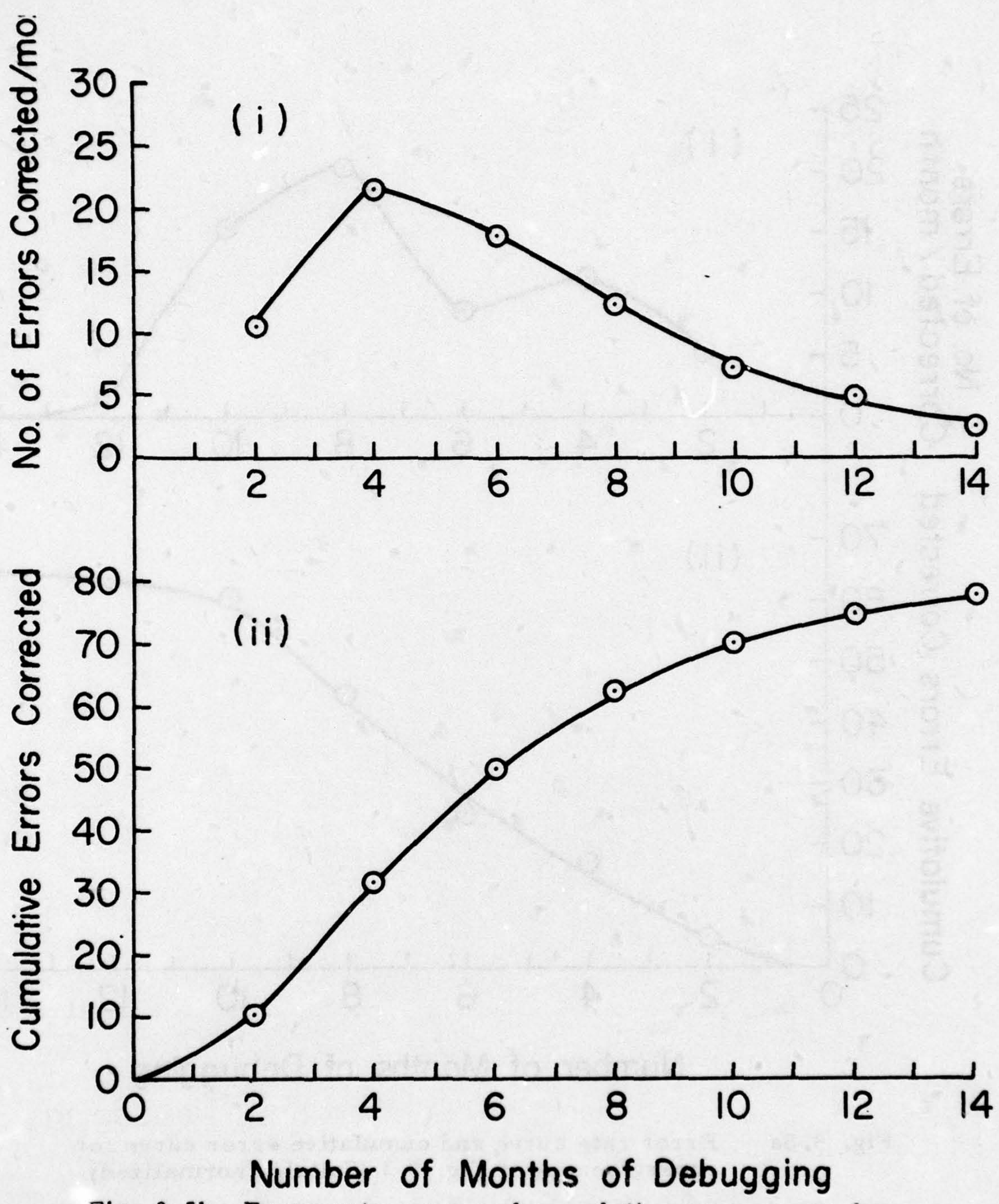


Fig. 3.5b Error rate curve and cumulative error curve for supervisory B of Fig. 3.1 (suitably normalized)

(i) Envelope of the error rate curve for Supervisory B
(ii) Cumulative error curve for Supervisory B (see Reference 1)

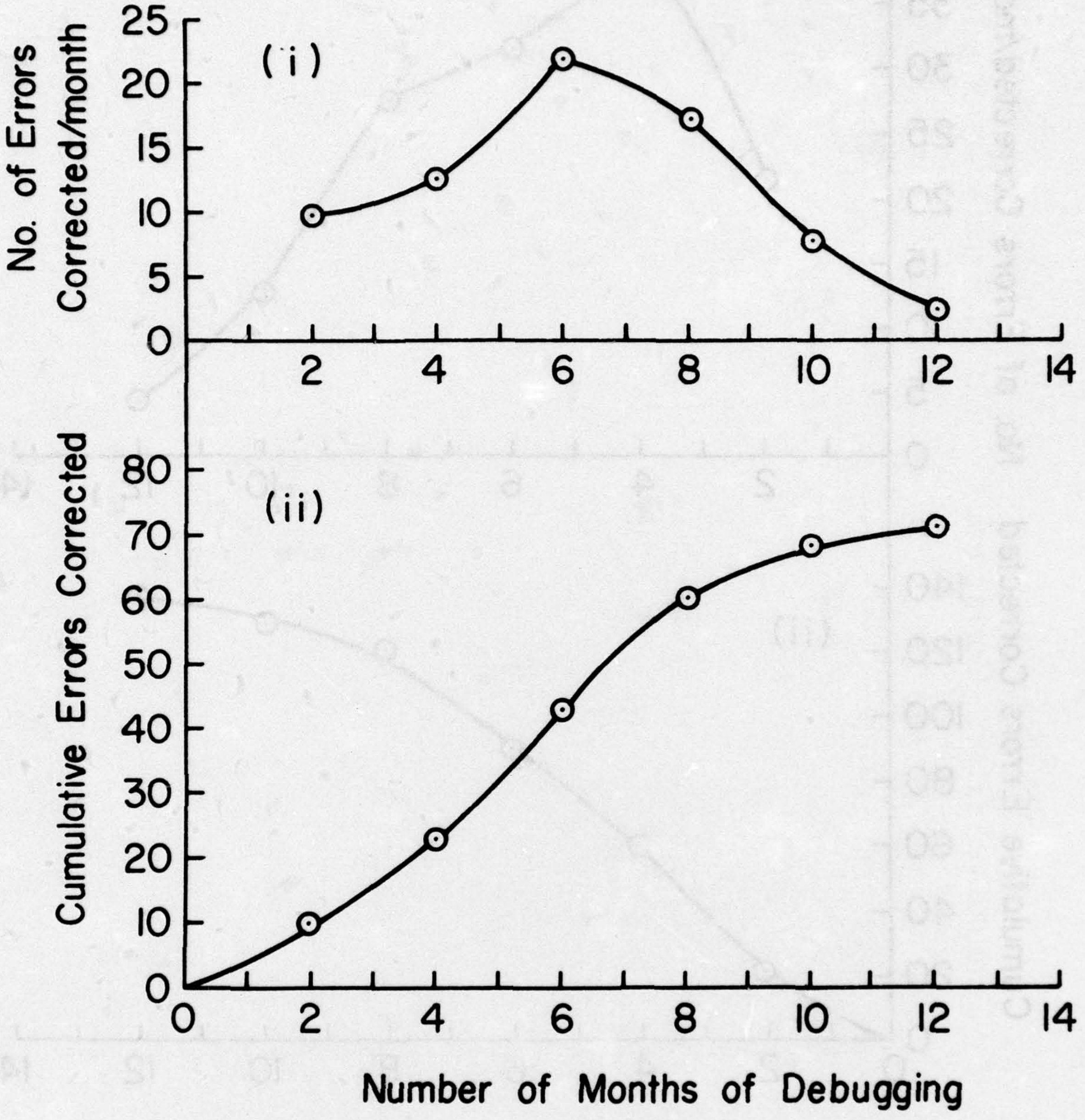


Fig. 3. 5c Error rate curve and cumulative error curve for Supervisory C of Fig. 3.1 (suitably normalized)

(i) Envelope of the error rate curve for Supervisory C
(ii) Cumulative error curve for Supervisory C (see Reference 1)

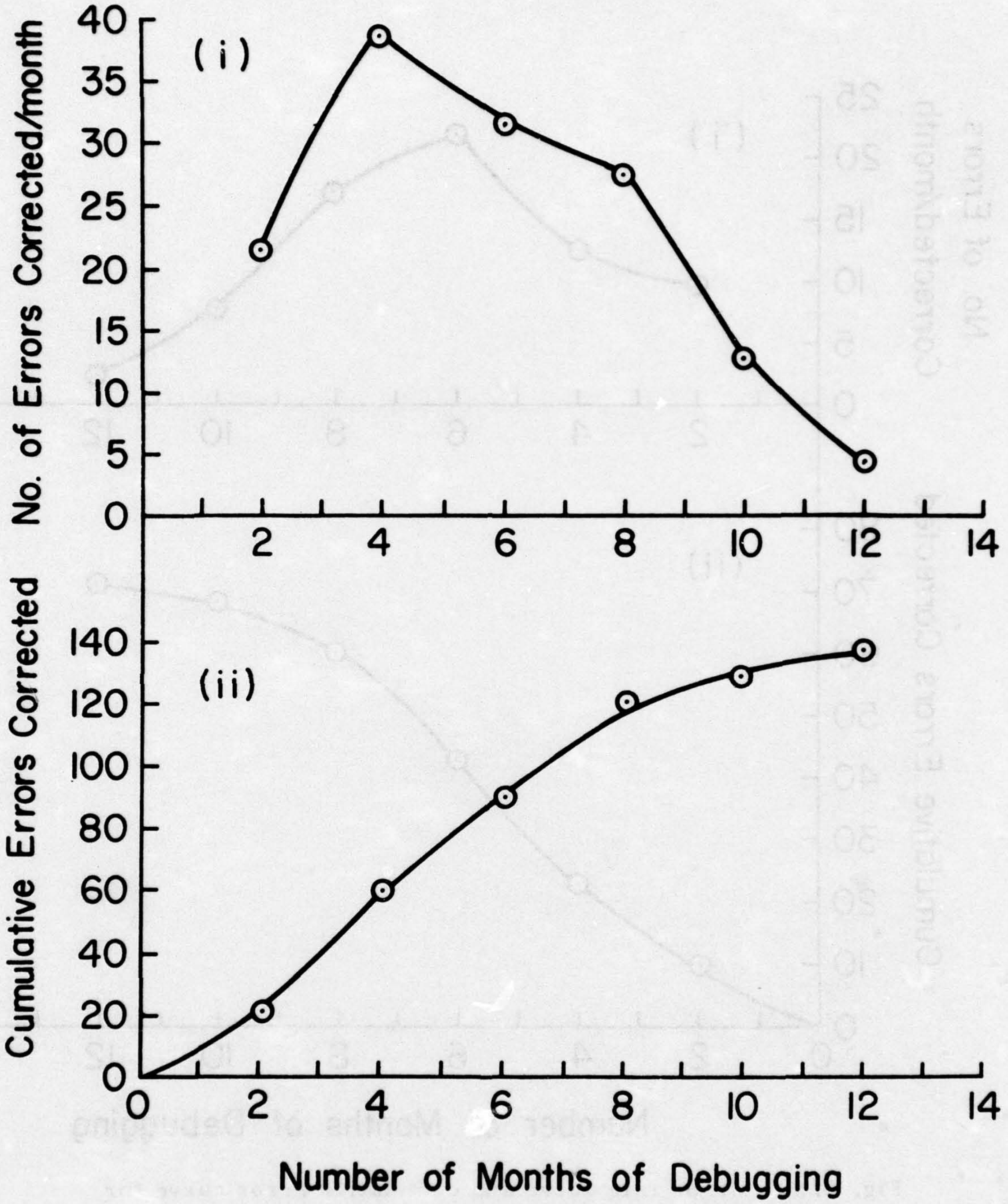


Fig. 3.5d Error rate curve and cumulative error curve for Application A of Fig. 3.2 (suitably normalized)

(i) Envelope of the error rate curve for Application A
(ii) Cumulative error curve for Application A (see Reference 1)

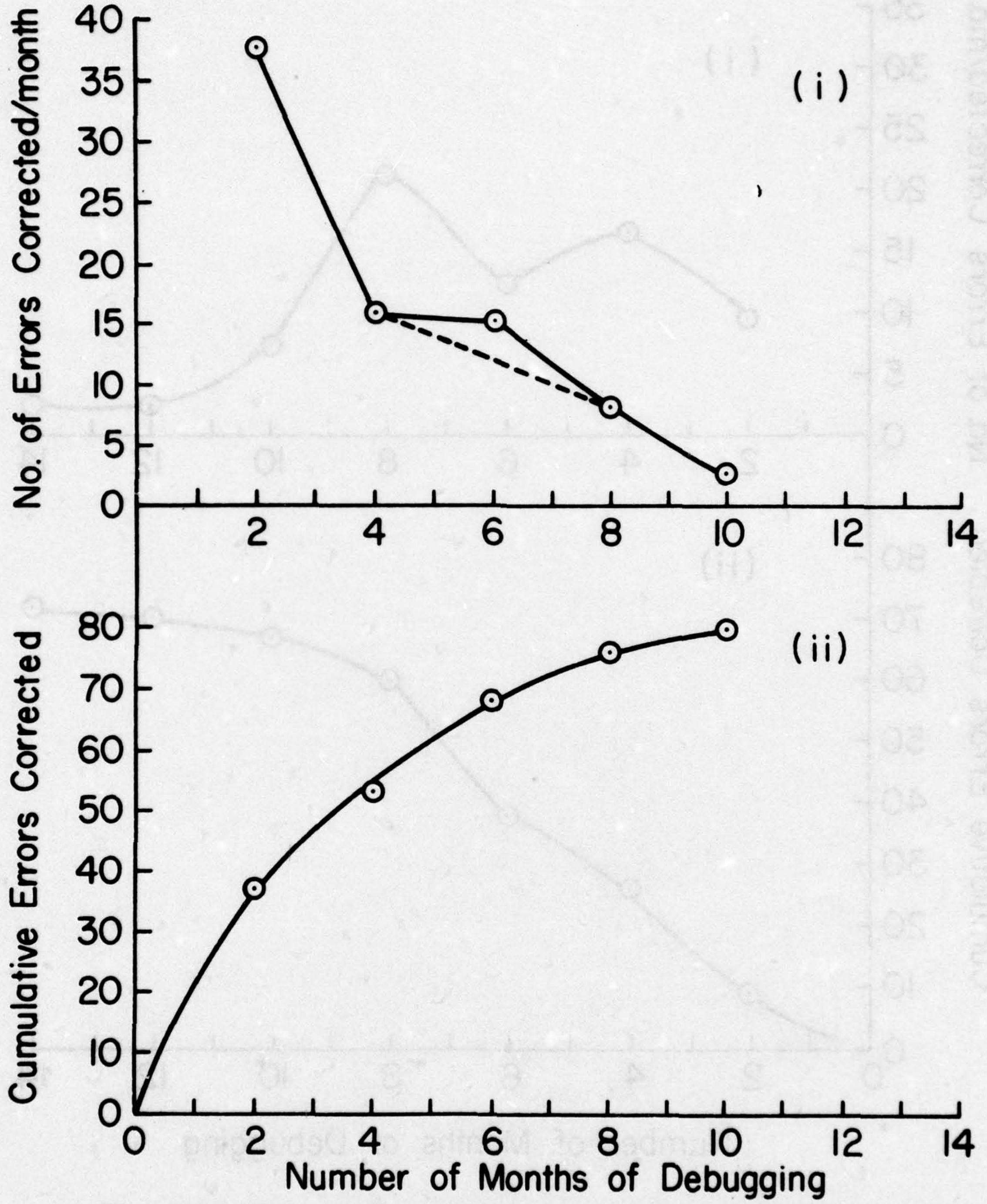


Fig. 3. 5e Error rate curve and cumulative error curve for Application B of Fig. 3. 2 (suitably normalized)

(i) Envelope of error rate curve for Application B
(ii) Cumulative error curve for Application B (see Reference 1)

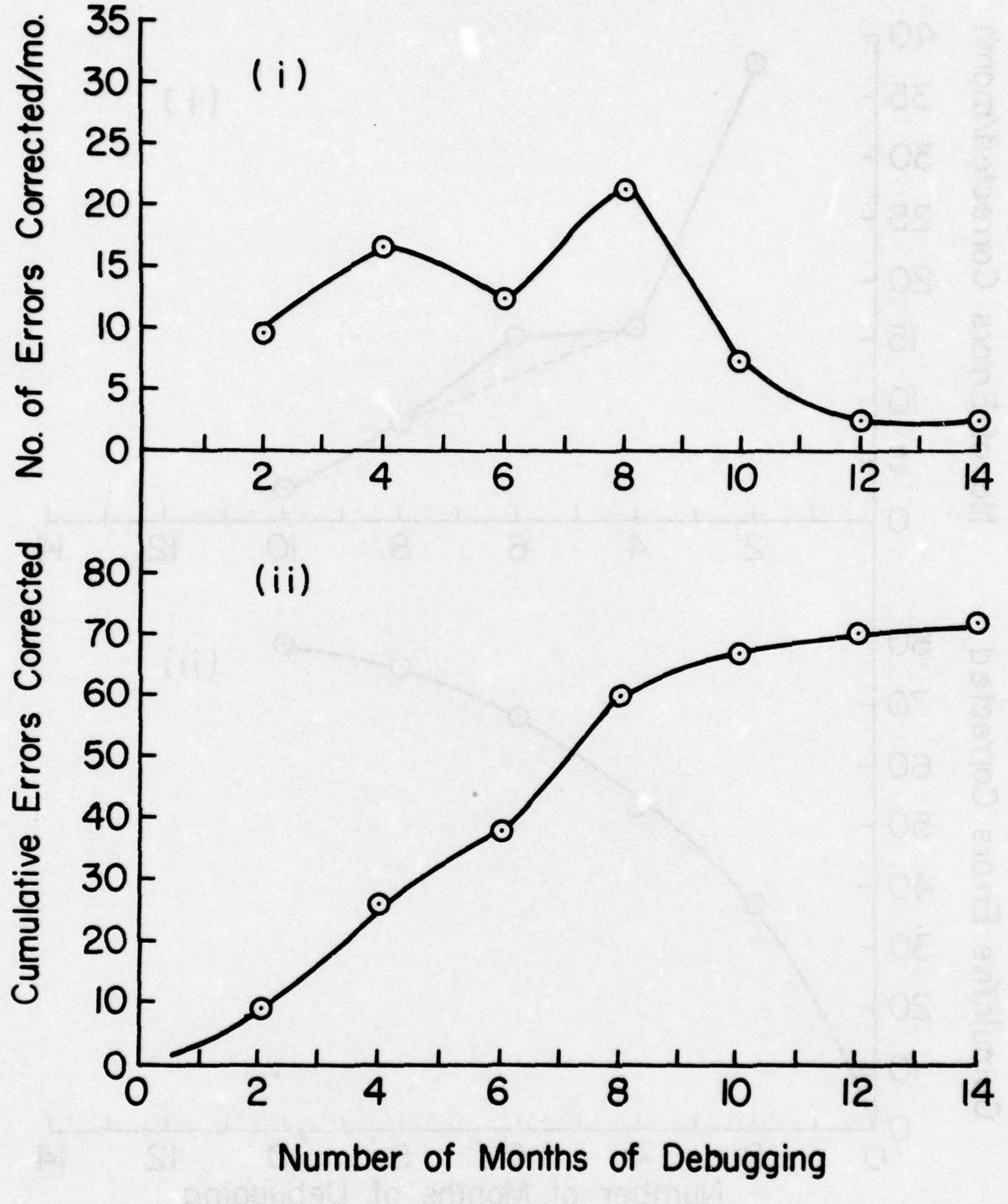


Fig. 3.5f Error rate curve and cumulative error curve for Application C of Fig. 3.2 (suitably normalized)

(i) Envelope of the error rate curve for Application C
(ii) Cumulative error curve for Application C (see Reference 1)

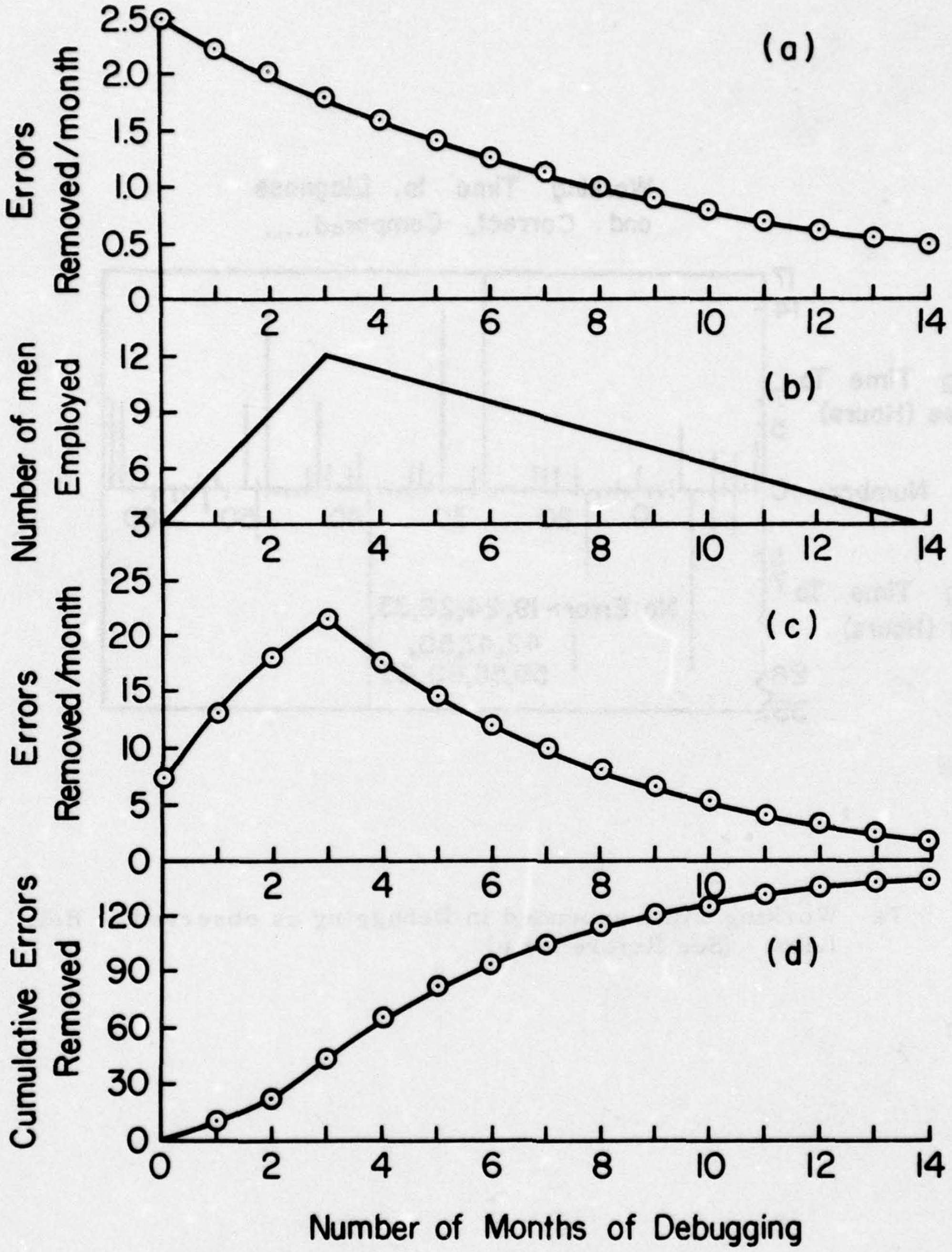


Fig. 3.6 Hypothetical error model based on the profile of curves in Figs. 3.3 and 3.5
(a) Variation of error removal per man-month over the debugging time
(b) Variation of manpower over the debugging time
(c) Variation of error removal per month over the debugging time
(d) Cumulative error removal

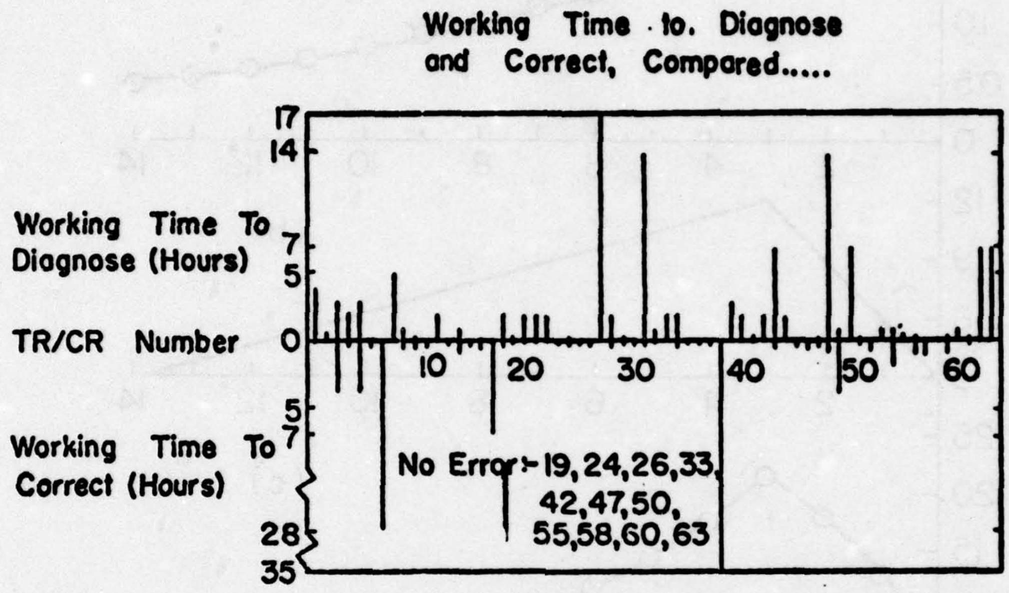


Fig. 3.7a Working Time expended in Debugging as observed at Bell Labs. (See Reference 6)

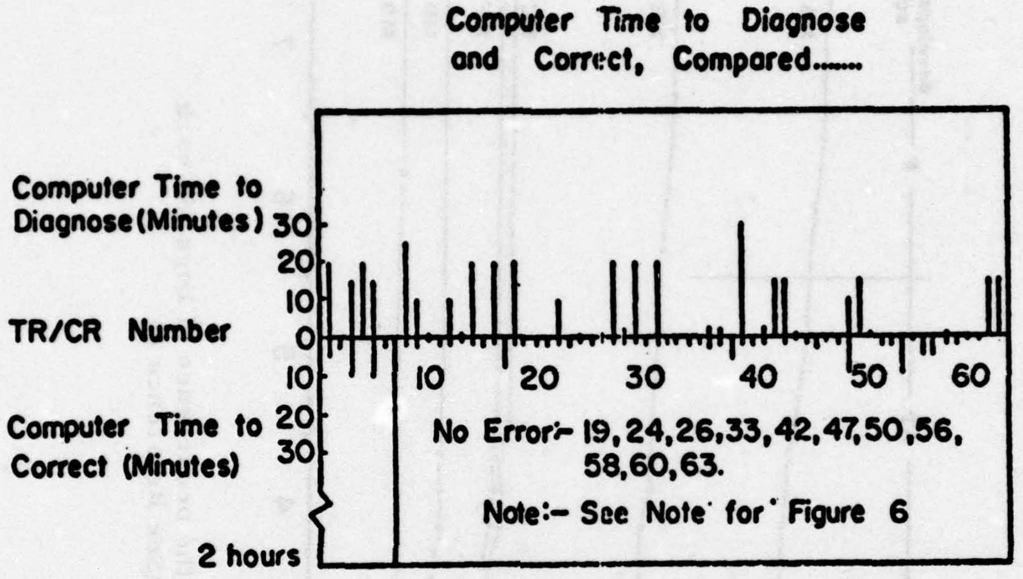


Fig. 3.7b Computer Time expended in Debugging as observed at Bell Labs. (See Reference 6)

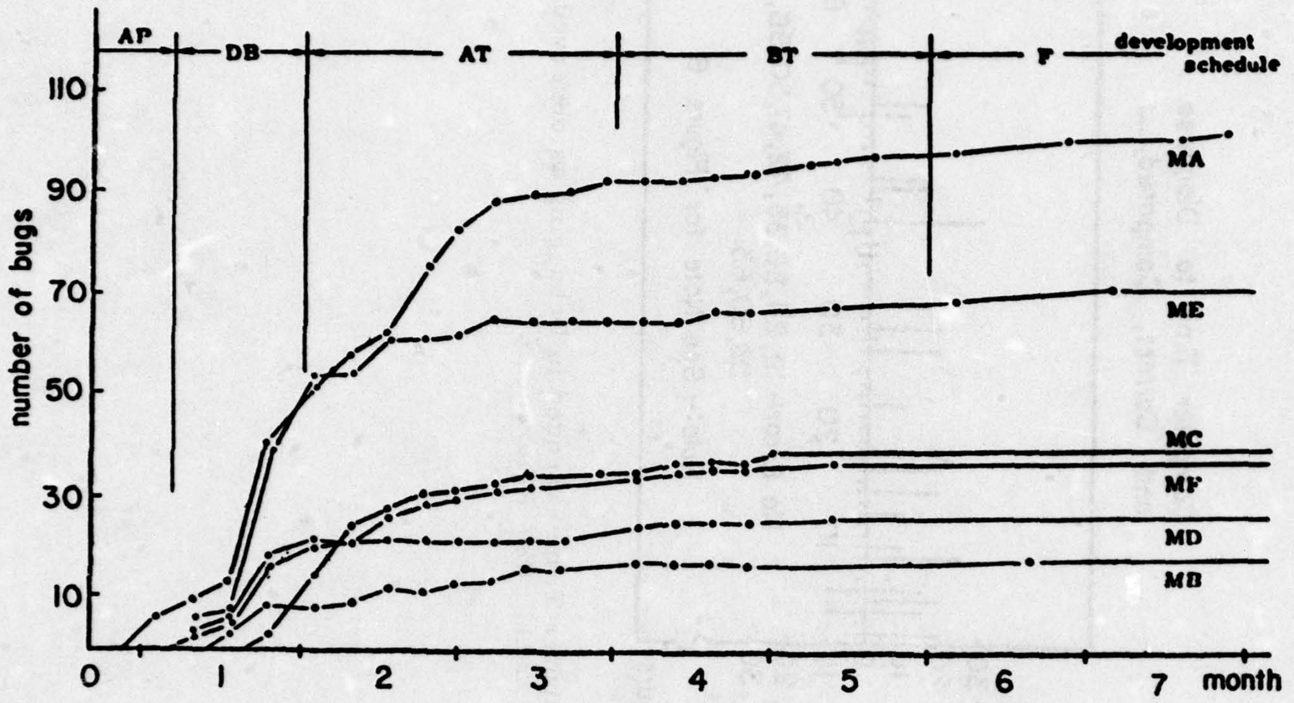


Fig. 3.8 Cumulative curves on the occurrence of bugs for each module of SAMPLE. (See Reference 8)

CHAPTER 4
ERROR GENERATION MODEL

4.1 Introduction

In Chapter 2 we discussed an error model in which the total number of program errors remained constant. At this point we examine the assumption that it is feasible to debug without generating an error. Most software personnel agree that there is a certain generation of errors associated with correction. The ways in which errors are generated was discussed in Chapter 1.

Depending upon the efficiency of debugging, the error generation rate could be greater than, equal to or less than the error correction rate. The resulting error behavior in the three cases is shown qualitatively in Fig. 4.1. (Figure 4.1a is a particular case where there is no generation of errors) [3]. The time τ_1 is when debugging stops. Note that the number of errors remaining > 0, in each case.

4.2 Model for Generated Errors

We wish to develop a set of equations which will describe the above cases. We begin by writing a difference equation for the number of errors in the program.

Errors present at time τ_i = [Errors present at time τ_{i-1}]
+ [Errors generated in the interval $(\tau_i - \tau_{i-1})$]
- [Errors removed in the interval $(\tau_i - \tau_{i-1})$].

If we let

$n_g(\tau_i, \tau_{i-1})$ = number of errors generated in the interval $(\tau_i - \tau_{i-1})$

$n_d(\tau_i, \tau_{i-1})$ = number of errors detected in the interval $(\tau_i - \tau_{i-1})$

$n_c(\tau_i, \tau_{i-1})$ = number of errors corrected in the interval $(\tau_i - \tau_{i-1})$

then the number of errors remaining in the program at time τ_i, $n(\tau_i)$, is given by the following difference equation

$$n(\tau_i) = n(\tau_{i-1}) + n_g(\tau_i, \tau_{i-1}) - n_c(\tau_i, \tau_{i-1}) \tag{4.1}$$

Conversion of the above difference equation to a differential equation is performed by grouping terms, dividing both sides by $(\tau_i - \tau_{i-1}) \equiv \Delta\tau$ and taking limits

$$\lim_{\Delta\tau \to 0} \left\{ \frac{n(\tau_i) - n(\tau_{i-1})}{\Delta\tau} \right\} = \lim_{\Delta\tau \to 0} \left\{ \frac{n_g(\tau_i, \tau_{i-1})}{\Delta\tau} \right\} - \lim_{\Delta\tau \to 0} \left\{ \frac{n_c(\tau_i, \tau_{i-1})}{\Delta\tau} \right\} \tag{4.2}$$

The left-hand side of Eq. (4.2) is recognized as the rate of change of errors remaining, i.e., the derivative of n with respect to τ. The right-hand side is composed of two terms, which as the limit is approached become the rates of error generation and correction respectively. The notation for error rates is given below

$r_g(\tau_i)$	$\triangleq$	Generation rate of new errors at time τ_i
$r_c(\tau_i)$	$\triangleq$	Correction rate of errors at time τ_i
$r_d(\tau_i)$	$\triangleq$	Detection rate of errors at time τ_i

Using the above definitions, Eq. (4.2) becomes

$$\frac{dn(\tau)}{d\tau} = r_g(\tau) - r_c(\tau) \tag{4.3}$$

In Eq. (4.3) the term accounting for error generation includes all the five cases discussed in Chapter 1 which are created by debugging changes. In a process where debugging is efficient, the generation rate is smaller than the correction rate and the number of bugs in the system decreases. A steady state is reached when correction decreases so that $dn(\tau)/d\tau = 0$. (See Fig. 4.1a and b) Let us hypothetically say that the correction rate is proportional to the detection rate.

$$r_c(\tau) = \beta\, r_d(\tau) \tag{4.4}$$

In the ideal case, all detected errors are immediately corrected and $\beta = 1$. However, in practice $0 < \beta < 1$ because some detected errors may be incorrectly fixed. The latter effect adds to the generation as previously discussed.

The generation effect is more complicated to model. If we assume tha the generation rate is proportional to correction rate, then

$$r_g(\tau) = \alpha\, r_c(\tau) \tag{4.5}$$

Combining Eqs. (4.3), (4.4) and (4.5) yields

$$\frac{dn(\tau)}{d\tau} = \alpha\, r_c(\tau) - \beta\, r_d(\tau) = \beta(\alpha - 1)\, r_d(\tau) \tag{4.6}$$

Inspection of Eq. (4.6) shows that the effect under these assumptions for the *normal* case where $\alpha < 1$ is that the number of errors decreases.

However, if $\alpha > 1$, generation exceeds correction and the result illustrated in Fig. 4.1c is obtained. If $\alpha = 0$, there is no generation, but in general $0 < \alpha < 1$ and the effect of error generation is to reduce the effective correction rate (see case 1, Appendix I). From the solution we can see that for large τ, $n(\tau)$ becomes negative which leads to a physical contradiction.

We may explore a different model by assuming the detection rate is in turn proportional to the number of remaining errors. This leads to Case 3 of Appendix I, and $n(\tau)$ is a decreasing exponential. This result agrees with only a part of the experimental data reported in the literature [1, 8], since shapes other than decreasing exponentials have also been observed for $n(\tau)$.

A different hypothesis is to assume that generation of new errors is a function of not only the detection rate, but also the number of remaining errors. Clearly, the larger the number of detected errors, the more are the changes required and the probability of creating an error increases. Also each time we make a change there is a chance that this will interact with existing errors and create new errors via the interaction. Assuming generation is a simple product of these two functions and correction is as given in Eq. (4.4) (with β replaced by b) we obtain

$$r_g(\tau) = a\, n(\tau)\, r_d(\tau) \tag{4.7}$$

$$r_c(\tau) = b\, r_d(\tau) \tag{4.8}$$

where a and b are proportionality constants. Substituting Eqs. (4.7) and (4.8) into Eq. (4.3) we obtain

$$\frac{dn(\tau)}{d\tau} = a\, n(\tau)\, r_d(\tau) - b\, r_d(\tau) \tag{4.9}$$

If we make the further assumption that the detection rate is a constant, r_o, Eq. (4.9) becomes

$$\frac{dn(\tau)}{d\tau} = a\, r_o\, n(\tau) - b\, r_o \tag{4.10}$$

The above differential equation can be readily solved by taking Laplace transforms or by classical differential equation theory yielding

$$n(\tau) = (n_o - b/a) \exp(a\, r_o \tau) + b/a \tag{4.11}$$

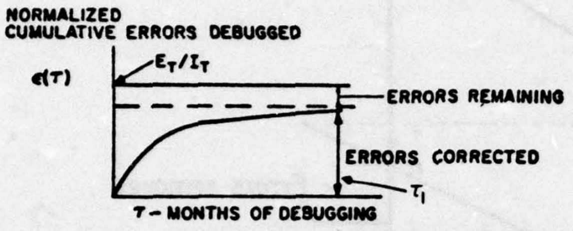
where $n(\tau = 0) \equiv n_o$.

The behavior of Eq. (4.11) depends upon the relative values of n_o and b/a. Since the probability of n_o being exactly equal to b/a is very low, we are left essentially with two possibilities. If $n_o > b/a$, then $n(\tau)$ builds up exponentially. If $n_o < b/a$, the number of errors decreases and becomes negative for large τ. The two cases are illustrated in Fig. 4.2, and Case 4, Appendix I.

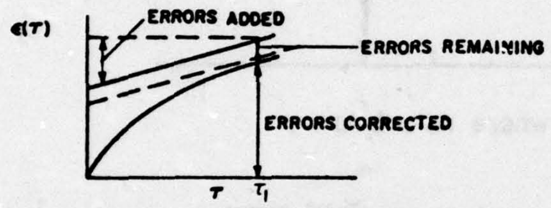
4.3 Contradictions and Basis for a Reformulated Model

A negative number of errors has no physical meaning, thus some cases of the models given in Table A-1 lead to a contradiction. Either the initial assumption and the model are only valid for a short time period or our initial assumptions were wrong. In fact we have assumed several models for error generation and have rejected the results (Eqs. 4.6 and 4.11) because the remaining errors went negative or the cumulative error correction curve did not resemble those observed experimentally. We now turn toward models for the error correction rate which may be more realistic, so as to obtain an expression for $n(\tau)$ which does not violate physical reasoning and fits the experimental data.

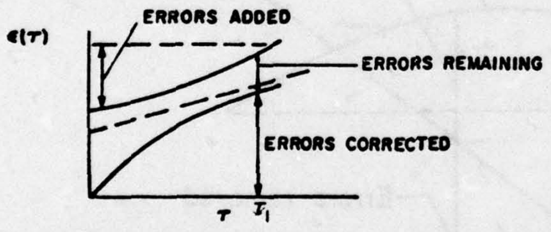
Programmers have reported that initially the bugs are easily removed. Only in the advanced stage of debugging, does the error removal become intricate. Although Shooman and Bolsky [6] present evidence that the effort to fix a bug is the same for early and later bugs, they did not feel the result was in itself conclusive enough to overturn the intuitive hypothesis that the later bugs are the hard ones to fix. In Table 3.1 and Figs. 3.1 and 3.2 we observe a decreasing trend in the correction rate with debugging time. Based on this fact if we assume that correction rate is proportional to the number of remaining errors, then this leads to a cumulative correction curve which does not conform to experimental results. A more complex model with features from both of the two extreme possibilities is developed in the next chapter.



(a) APPROACHING EQUILIBRIUM, HORIZONTAL ASYMPTOTE, NO GENERATION OF NEW ERRORS.

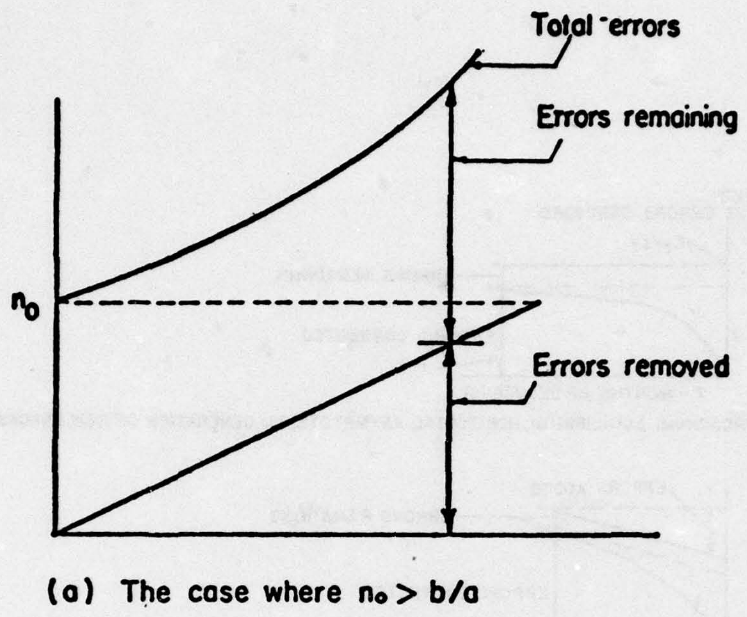


(b) APPROACHING EQUILIBRIUM, GENERATION RATE OF NEW ERRORS EQUALS ERROR REMOVAL RATE.



(c) DIVERGING PROCESS, GENERATION RATE OF NEW ERRORS EXCEEDS ERROR REMOVAL RATE.

Fig. 4.1 Cumulative errors debugged versus months of debugging. (See Reference 3)



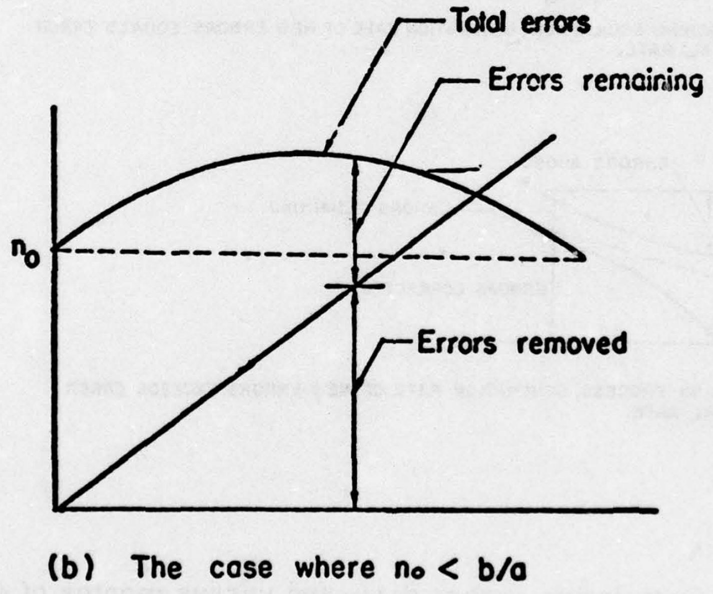


Fig. 4.2 The model developed under the assumptions of Case 4 Table A-1. (See Reference 4)

CHAPTER 5

ERROR CORRECTION MODEL

5.1 Introduction

In Chapter 4 we found that our assumption of constant error correction rate resulted in the number of remaining errors eventually going negative. This leaves us with several alternatives.

1. We can discount the results of the experiment conducted at Bell Labs [6]

2. We can accept the earlier hypothesis that the correction rate is proportional to the number of remaining errors

3. We could formulate other hypotheses.

There is no definitive answer to these questions. We could argue against the experiment conducted at Bell Labs [6] in two ways. On the one hand this conclusion was based on the results of just one experiment conducted on a program which can be considered as between a small and medium sized program. Secondly, the reference did not discuss generation rate of errors. In Chapter 4 we attempted a few models for error generation, Eqs. (4.6) and (4.11), and found the results *unsatisfactory*. We now turn our attention toward other models for error correction.

5.2 A Two-Phase Model

In earlier work [1, 8], a decreasing trend in error correction rate has been observed towards the later part of debugging. Figures 3.3, 3.4 and 3.8 illustrate this fact. On the other hand, Shooman and Bolsky [6] observed a constant correction rate. Also, discussions with program managers have led to the conclusion that correction rate is sometimes manpower limited. Thus, we postulate a new model where the correction rate remains constant during the early stage of debugging (manpower limited). We assume that later in the program another stage is reached where the correction rate is proportional to the number of remaining errors. During both these correction stages we assume the error generation rate is proportional to the product of the number of remaining errors and the number of detected errors as in Eq. (4.7). The transition from the early stage model of error correction to the later stage model may be considered to occur at a critical value of the remaining number of bugs which we call n_1. These assumptions lead to

$$r_g = p_1\, n(\tau)\, r_d(\tau) \qquad \text{for all } n(\tau) \qquad (5.1)$$

$$r_c = p_2\, r_d(\tau) \qquad \text{for all } n(\tau) > n_1 \text{ [Region 1]} \qquad (5.2a)$$

$$= p_3\, n(\tau)\, r_d(\tau) \qquad \text{for } n(\tau) \leq n_1 \text{ [Region 2]} \qquad (5.2b)$$

where p_1, p_2, p_3 are constants of proportionality.

If we make the further assumption that $r_d(\tau)$ is a constant we obtain

$$r_g = a_1 \, n(\tau) \tag{5.3}$$

$$r_c = k_1 \qquad \text{for } n(\tau) > n_1 \tag{5.4a}$$

$$\quad = k_2 \, n(\tau) \qquad \text{for } n(\tau) \leq n_1 \tag{5.4b}$$

where

$$a_1 = p_1 \, r_d(\tau)$$

$$k_1 = p_2 \, r_d(\tau)$$

$$k_2 = p_3 \, r_d(\tau)$$

5.3 A Manpower Limited Model

The model which was developed in the previous section retains the assumption made in Chapter 4 that the error generation rate is proportional to the number of errors remaining [c.f., Eq. (5.3)]. However, the correction rate is governed by Eqs. (5.4a and b). Rewriting Eq. (4.3) for convenience we have

$$\frac{dn(\tau)}{d\tau} = r_g(\tau) - r_c(\tau) \tag{5.5}$$

Substituting Eqs. (5.3) and (5.4a) into Eq. (5.5) we get for the early phase where $n(\tau) > n_1$ (called Region 1)

$$\frac{dn(\tau)}{d\tau} = a_1 \, n(\tau) - k_1 \tag{5.6}$$

The solution of Eq. (5.6) is accomplished in a manner similar to Eq. (4.11) yielding

$$n(\tau) = (n_o - k_1/a_1)\, e^{a_1 \tau} + k_1/a_1 \tag{5.7}$$

where

$$n(\tau = 0) = n_o$$

If $n_o > k_1/a_1$ the debugging is out of control and Eq. (5.7) indicates that the errors build up exponentially with time.* On the other hand if $n_o < k_1/a_1$,

*Discussion with experienced software managers have verified that on rare occasions debugging does go out of control and either the program is scrapped and rewritten or a new team of 'Super Debuggers' is brought in.

the correction process is efficient and the errors reduce with time. Once the errors fall to the critical value n_1, a transition takes place in the error correction rate and substitution of Eqs. (5. 3) and (5. 4b) into Eq. (5. 5) yields

$$\frac{dn(\tau)}{d\tau} = a_1 n(\tau) - k_2 n(\tau) \tag{5.8}$$

Letting the time elapsed in reducing the number of errors to n_1 be τ_1, a solution of Eq. (5. 8) yields

$$n(\tau) = n_1 \exp[(a_1 - k_2)(\tau - \tau_1)] \tag{5.9}$$

The conditions under which the transition occurred are explained as follows: As the errors decrease the number of men employed is also reduced [9]. Since we intuitively feel that the later bugs are harder to fix, even if we maintain the ratio of the number of errors to number of men constant we will observe a decreasing correction rate. (Note, this contradicts some of the results of Ref. 6.) We have assumed an abrupt change in the number of men employed to reduce mathematical complexity. In practice the change is gradual.

Eq. (5. 9) in itself gives rise to two cases depending upon whether $1 > k_2/a_1$ or $1 < k_2/a_1$. If $1 < k_2/a_1$, then $n(\tau)$ decays exponentially to zero as τ goes to infinity. On the other hand, if $1 > k_2/a_1$ then $n(\tau)$ increases exponentially, thereby re-entering Region 1. A physical explanation for why the switch in regions may occur is that the quantity k_2 is associated with the reduction in manpower. As testers are removed from the project, k_2 decreases. If these testers are removed prematurely, k_2 may decrease to the extent that $1 > k_2/a_1$ thereby causing a re-entry into Region 1. With increasing errors some personnel are brought back and a transition to Region 2 will soon occur. We thus can have an oscillating model. It is convenient to categorize the model as

<u>Case 1</u> The Unstable Model

<u>Case 2</u> The Controlled Model

<u>Case 3</u> The Oscillatory Model

The three cases are illustrated qualitatively in Fig. 5. 1 and summarized in Case 5, Appendix I. We will analyze each case separately.

<u>Case 1</u>

In Fig. 5. 1a the errors are seen to diverge exponentially. If the software personnel are not sufficiently experienced, a more experienced team could be brought in to replace the current one. At this point let us say, the errors have built up to n' at time τ'. The new team establishes different values for k_1 and a_1 (viz. k_1' and a_1') such that $n' < k_1'/a_1'$.

The equation describing error correction is

$$n_c(\tau) = k_1 \tau \quad \text{for} \quad \tau < \tau' \tag{5.10}$$

while $n(\tau)$ is described by Eq. (5.7). The total number of errors at any time is the sum of those remaining and those corrected, resulting in

$$n_t(\tau) = n(\tau) + n_c(\tau) \tag{5.11}$$

For $\tau > \tau'$ the error equation can be written using Eq. (5.7) resulting in

$$n(\tau) = (n' - k_1'/a_1') \exp[a_1'(\tau - \tau')] + k_1'/a_1' \tag{5.12}$$

The error correction is given by

$$n_c(\tau) = k_1\tau' + k_1'(\tau - \tau') \tag{5.13}$$

Equations (5.12) and (5.13) are applicable between τ' and τ_1, where τ_1 is the instant of time at which the errors fall to the critical value n_1. Beyond τ_1 the error equation is defined by Eq. (5.9). To compute the number of errors corrected we can rewrite Eq. (5.4b) as

$$\frac{dn_c(\tau)}{d\tau} = .k_2\, n(\tau) \text{ which results in}$$

$$n_c(\tau) = \int k_2\, n(\tau)\, d\tau$$

or

$$n_c(\tau) = k_2 n_1 \int \exp[(a_1' - k_2)(\tau - \tau_1)]\, d\tau$$

or

$$n_c(\tau) = \frac{k_2 n_1}{a_1' - k_2} \exp[(a_1' - k_2)(\tau - \tau_1)] + C \tag{5.14}$$

Using the boundary condition that $n_c(\tau_1) = k_1\tau' + k_1'(\tau_1 - \tau')$ which is true from Eq. (5.13) the number of corrected errors in Eq. (5.14) may be written down as

$$n_c(\tau) = k_1\tau' + k_1'(\tau_1 - \tau') + \frac{k_2 n_1}{a_1' - k_2} \{\exp[(a_1' - k_2)(\tau - \tau_1)] - 1\} \tag{5.15}$$

Case 2

The initial stage here is described by Eq. (5.7) with $n_o < k_1/a_1$. The correction is governed by Eq. (5.10) except for the fact that the equation holds till time τ_1. The errors fall exponentially and a transition occurs at τ_1 beyond which the error model is described by Eq. (5.9), with $1 < k_2/a_1$. The correction curve beyond τ_1 is described by Eq. (5.14) with only a difference of a_1 replacing a_1' and the boundary condition being

$$n_c(\tau_1) = k_1\tau_1$$

The correction curve is therefore described by

$$n_c(\tau) = k_1\tau_1 + \frac{k_2 n_1}{a_1 - k_2}\{\exp[(a_1-k_2)(\tau-\tau_1)] - 1\} \text{ for } \tau \geq \tau_1 \tag{5.16}$$

Case 3

In the case of the oscillatory model a substantial number of errors have been removed and if the oscillations can be broken the system debugging can be promptly completed. As explained earlier one of the reasons for k_2 being less than a_1 is that the reassignment of some of the debuggers was done prematurely. If the transition had occurred at a lower value of n, the pattern might have been the same as in Case 2. Therefore, once the process goes back to Region 1, the software manager may increase the debugging personnel. If this is effective we may stay in Region 2 till completion of debugging; however, if personnel are removed again we may restart the oscillatory behavior.

Here the initial behavior of errors is given by Eq. (5.7) with $n_o < k_1/a_1$ while the correction is described by Eq. (5.10). Upon a transition at $n = n_1$ the model obeys Eq. (5.9) with $1 > k_2/a_1$. After transition the correction is described by Eq. (5.16). $n(\tau)$ now increases to n_2 at time τ_2 at which point another transition (this happens due to personnel being brought back) occurs bending the error curve downward. Under these conditions the model is described by an equation similar to Eq. (5.7). The equation is

$$n(\tau) = (n_2 - k_1/a_1)\exp[a_1(\tau - \tau_2)] + k_1/a_1 \tag{5.17}$$

Using Eq. (5.16) the correction curve can now be written as

$$n_c(\tau) = k_1\tau_1 + \frac{k_2 n_1}{a_1 - k_2}\{\exp[(a_1-k_2)(\tau_2-\tau_1)] - 1\} + k_1(\tau-\tau_2) \tag{5.18}$$

$$\text{for } \tau \geq \tau_2$$

The errors now fall until a critical point defined by $n = n_3$*at τ_3 when another

*n_3 is set smaller than n_1 to avoid oscillations. Under these conditions Region 2 includes the zone where $n \leq n_3$. The software team now establishes a new value for k_2 (viz. k_3) such that $1 \leq k_3/a_1$.

transition takes place. Below n_3 the error equation is described by an equation similar to that of Eq. (5.9). The equation is

$$n(\tau) = n_3 \exp[(a_1 - k_3)(\tau - \tau_3)] \quad \text{for} \quad \tau \geq \tau_3 \tag{5.19}$$

Once again Eq. (5.16) can be used to write an equation which describes the correction beyond τ_3.

$$n_c(\tau) = \frac{k_3 n_3}{a_1 - k_3} \{\exp[(a_1 - k_3)(\tau - \tau_3)] - 1\} + n_c(\tau_3) \tag{5.20}$$

where

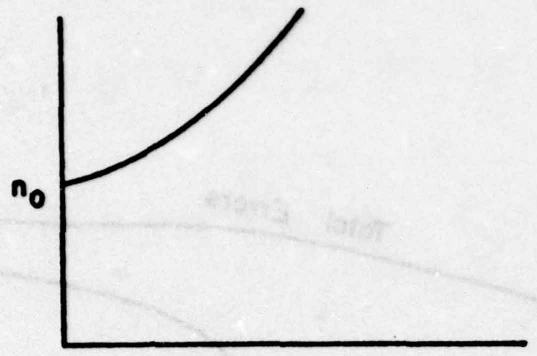
$$n_c(\tau_3) = k_1 \tau_1 + k_1(\tau_3 - \tau_2) + \frac{k_2 n_1}{a_1 - k_2} \{\exp[(a_1 - k_2)(\tau_2 - \tau_1)] - 1\}$$

which is obtained by substituting τ_3 for τ in Eq. (5.18). The three cases are illustrated in Figs. 5.2, 5.3 and 5.4. An initial error quantity of $n_o = 100$ has been assumed in all cases.

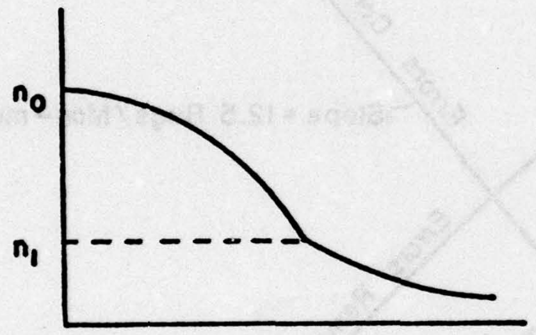
A PL/I computer program (see Appendix II for a listing and flowchart) has been written which plots the number of remaining errors, the cumulative corrected errors and the total number of errors. The calculated points for the curves in Figs. 5.2, 5.3 and 5.4 were computed using this program.

5.4 Summary

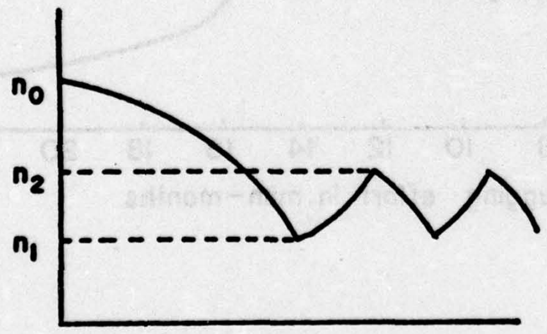
Many of the features of these models agree with the data reported in the literature and the experiences of program managers. Since basic data on manpower deployment, error rate, the rate of error generation, etc., are largely unavailable, testing the validity of the above models must await further experimental result. Assuming the above models are valid, we can investigate the economic constraints imposed upon Case 1. An economic break-even between rewriting the program and extensive debugging is discussed in the next chapter.



(a) Unstable Model. Errors build-up indiscriminately.



(b) Controlled Model. Debugging is efficient.



(c) Oscillatory Model.

Fig. 5.1 Remaining errors plotted as a function of months of debugging.

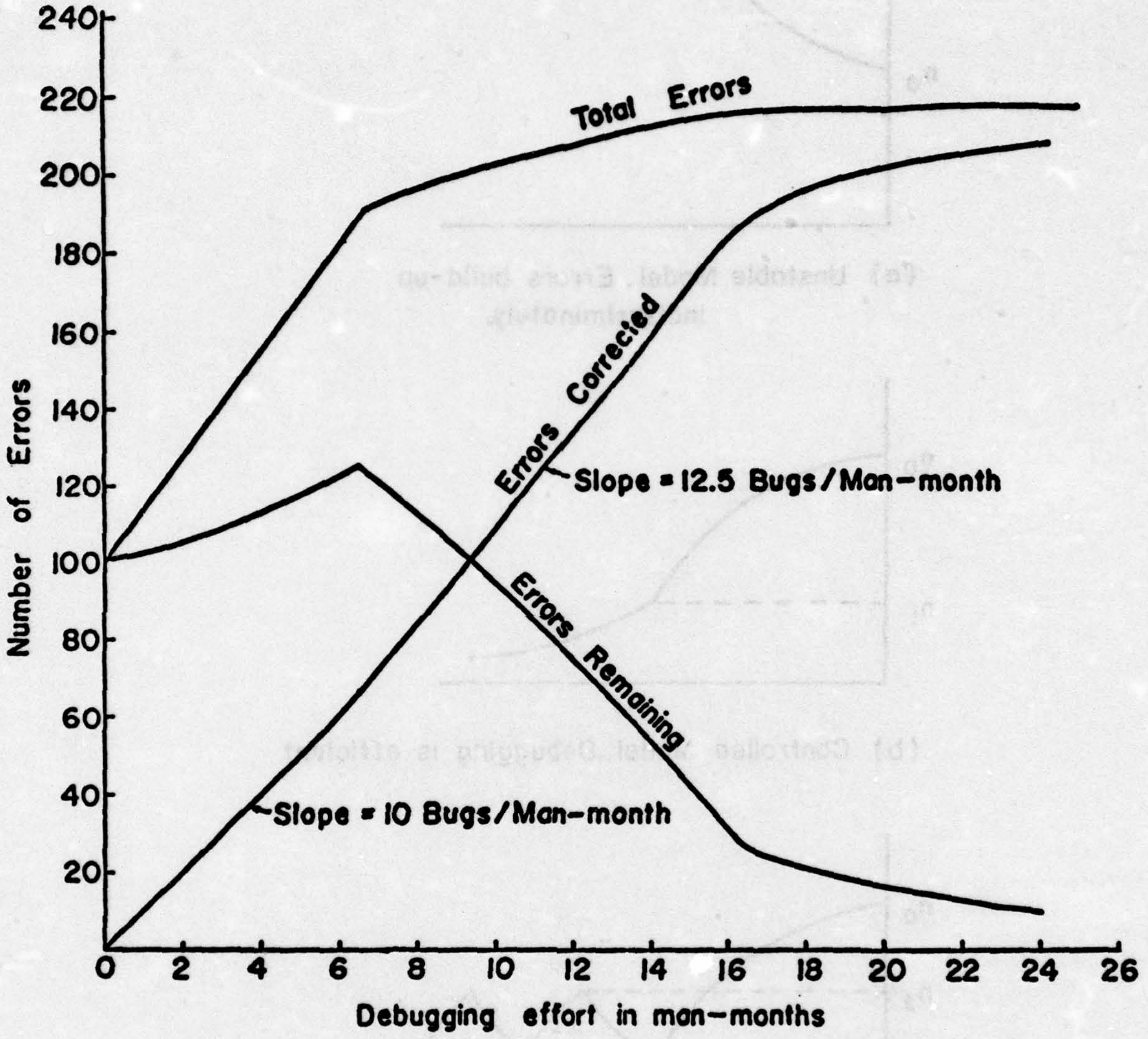


Fig. 5.2 Case 1 Unstable Model Controlled Subsequently. (See Eqs. 5.7, 5.9, 5.10, 5.11, 5.12, 5.13, 5.15)

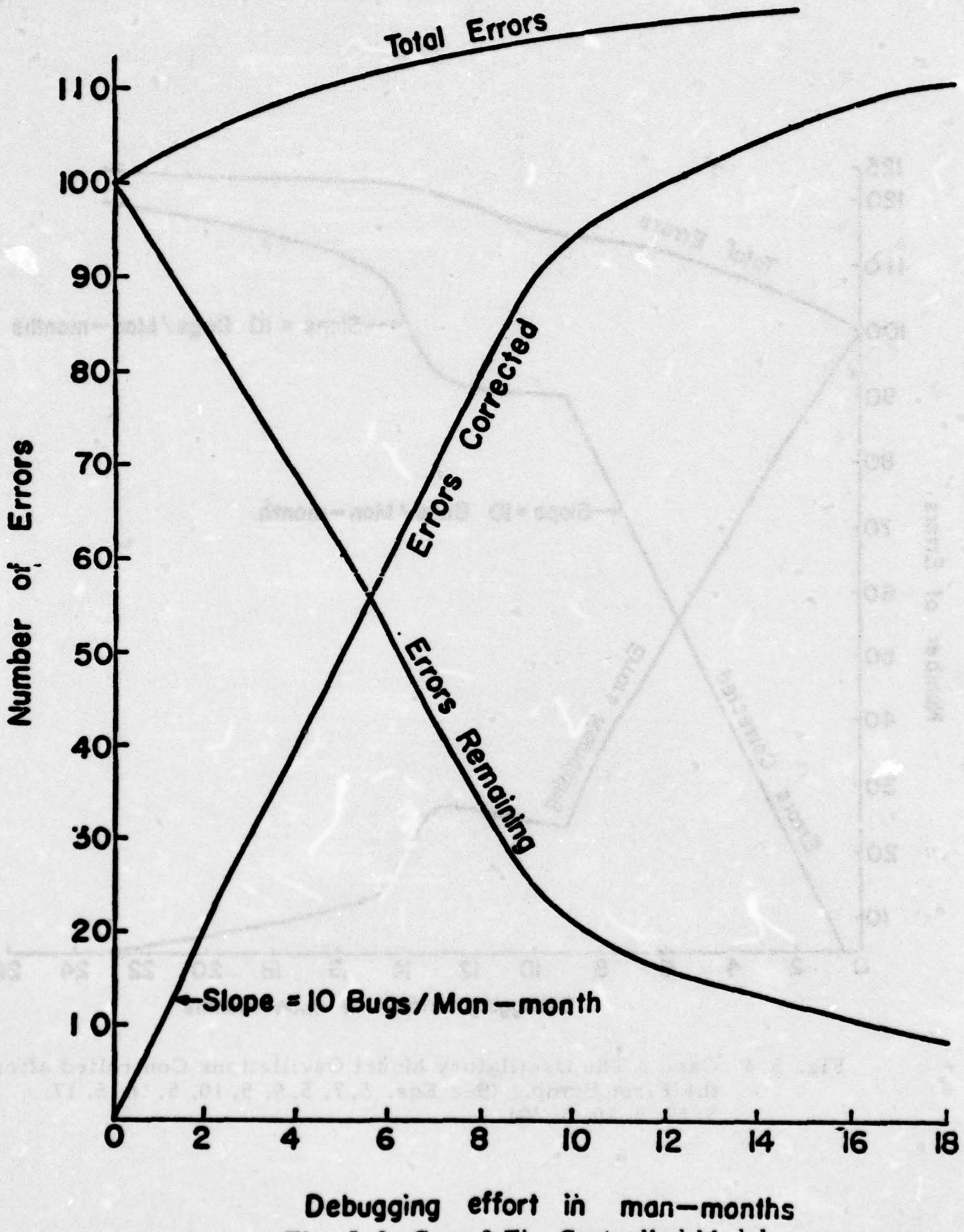


Fig. 5.3 Case 2 The Controlled Model. (See Eqs. 5.7, 5.9, 5.10, 5.16)

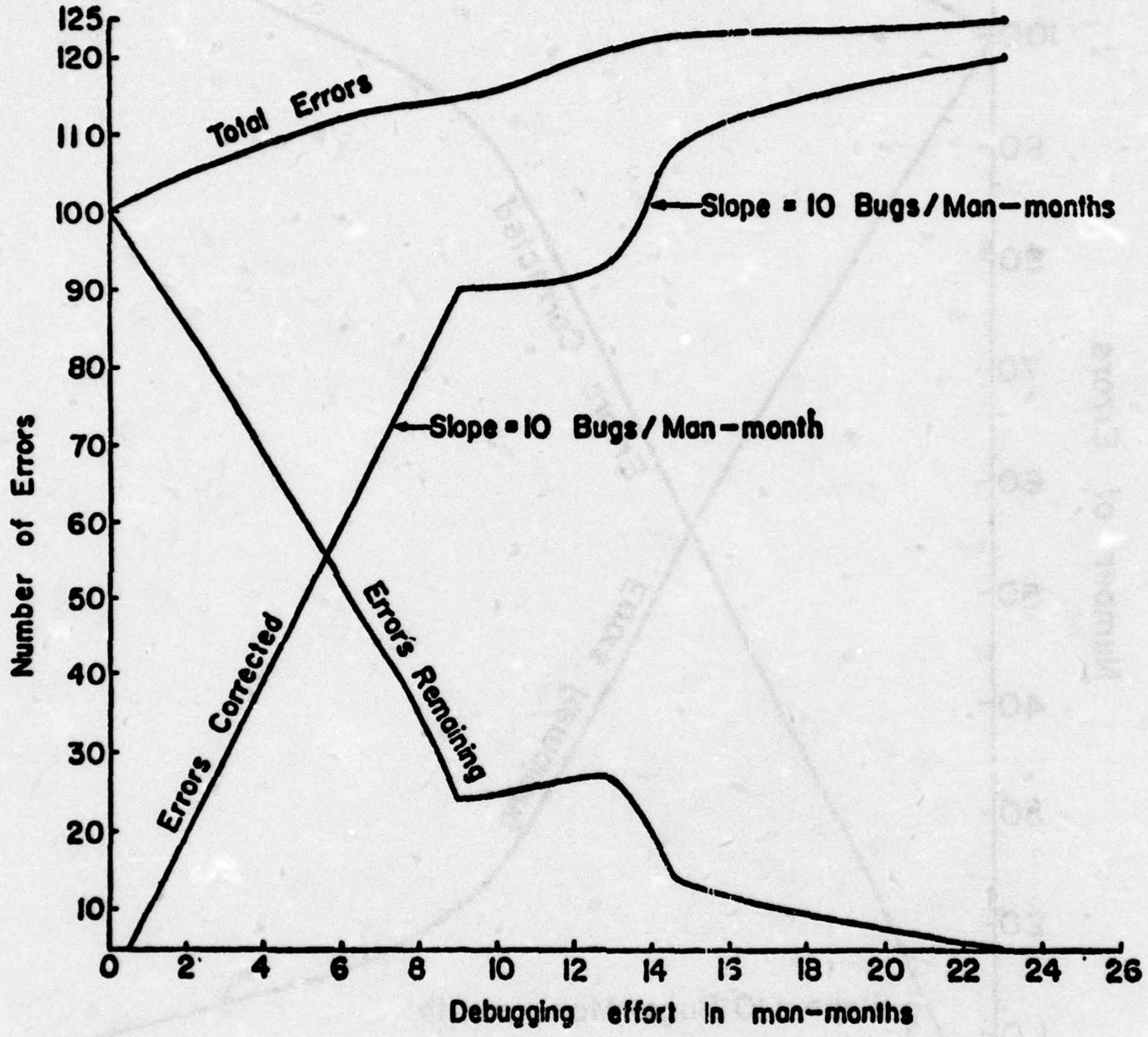


Fig. 5.4 Case 3 The Oscillatory Model Oscillations Controlled after the First Bump. (See Eqs. 5.7, 5.9, 5.10, 5.16, 5.17, 5.18, 5.19, 5.20)

CHAPTER 6

DEBUGGING COST MODELS

6.1 Introduction

The problem of computing an economic break-even point looks easy at first glance. However, it involves modeling of the penalty which the customer will demand for any delay caused in getting the system working.* Based on the models previously developed there is no closed form expression which can be devised to calculate compensation because there is no closed form expression for the length of the penalty period. This is clear if we examine the equation describing the error buildup (c.f. Eq. 5.7 with $n_o > k_1/a_1$). Since this equation is not linear, we have to compute the penalty period for every value of $n(\tau)$. However, one can perform a parametric study and produce tables and/or graphs for the penalty period.

6.2 Programming Economics Without Penalty

In Chapter 2, we stated that no program can be absolutely free of errors. In other words, bugs cannot be eliminated completely. Therefore, the economic model includes the effects caused by a trace of residual errors in the program. Although in most of the debugging experiments the errors do not build up initially (as depicted in Case 1, Chapter 5) we consider it to be true here and base our economic model on this extreme case. However, the model could be extended to cases where the buildup may occur sometime during the course of debugging. We assume that the errors build up from n_o (initial error content) to n'. After a transition at this point (which is the result of introduction of super-debuggers), the errors fall to n_1 where the next transition takes place (c.f. Eq. 5.9). The errors subsequently fall to a final value n_2 which is the residual number of errors in the program. The final value, n_2, which is achieved must result in a satisfactory level of operational reliability. With a knowledge of n', n_1 and n_2 one can solve Eqs. (5.9) and (5.12) and calculate the time required to achieve the level of satisfaction desired.

We are not interested in computing the expenses prior to time τ' (the instant of time at which $n(\tau)$ has built up to n') because our task is to compare the cost of rewriting the program with that of further debugging. The amount of money expended till τ' has no influence on the alternatives that will be resorted to. Evaluation of the time expended in debugging beyond τ' is carried out by solving Eq. (5.12) for τ. Since τ' has no significance except for evaluating the penalty which is explained later, we may move the coordinate axes to τ'. For an error content of n we get

$$\tau = \frac{1}{a_1'} \ln \left(\frac{n - k_1'/a_1'}{n' - k_1'/a_1'} \right) \tag{6.1}$$

*A contract with a penalty clause for late delivery explicitly states the delay penalty in dollars.

The time τ_1, for the errors to decrease from n' to n_1 is known from Eq. (6.1) by fixing $n = n_1$. Therefore

$$\tau_1 = \frac{1}{a'_1} \ell n \left(\frac{n_1 - k'_1/a'_1}{n' - k'_1/a'_1} \right) \tag{6.2}$$

After a transition at $\tau = \tau_1$ the error model is described by Eq. (5.9). Solving Eq. (5.9) for τ we get

$$\tau = \tau_1 + \frac{1}{(a'_1 - k_2)} \ell n \left(\frac{n}{n_1} \right) \tag{6.3}$$

By fixing $n = n_2$ (the residual number of bugs in the program) and substituting for τ_1 from Eq. (6.2), Eq. (6.3) may be rewritten as

$$\tau_2 = \frac{1}{a'_1} \ell n \left(\frac{n_1 - k'_1/a'_1}{n' - k'_1/a'_1} \right) + \frac{1}{(a'_1 - k_2)} \ell n \left(\frac{n_2}{n_1} \right) \tag{6.4}$$

where τ_2 is the time in man-months spent in reducing the number of errors from n' to n_2.

For simplicity we will first formulate a model which does not involve any penalty for delay. Subsequently we will include penalty terms for delay and arrive at a thorough analysis.

Let us say that the debugging cost per man-month is c_o. Thus the debugging expense, c_d, incurred in going from n' to n_2 is

$$c_d = c_o \tau_2 \tag{6.5}$$

If c represents the cost of rewriting, debugging, and testing the program then the break-even occurs when $c_d = c$. Substituting for τ_2 from Eq. (6.4) into Eq. (6.5) we get for break-even

$$c = c_o \left\{ \frac{1}{a'_1} \ell n \left(\frac{n_1 - k'_1/a'_1}{n' - k'_1/a'_1} \right) + \frac{1}{a'_1 - k_2} \ell n \left(\frac{n_2}{n_1} \right) \right\} \tag{6.6}$$

Solving Eq. (6.6) for n' we get

$$n' = n_1 \exp\left\{a_1'\left[\frac{1}{a_1'-k_2} \ell n\left(\frac{n_2}{n_1}\right) - \frac{c}{c_o}\right]\right\}$$

$$+ \frac{k_1'}{a_1'}\left\{1 - \exp\left[a_1'\left\{\frac{1}{a_1'-k_2} \ell n\left(\frac{n_2}{n_1}\right) - \frac{c}{c_o}\right\}\right]\right\} \qquad (6.7)$$

Using Eq. (6.7) a curve can be plotted with n' as a function of c/c_o. Since c is the total cost of rewriting and c_o is the debugging cost per man-month the ratio c/c_o is in man-months. Such a curve is portrayed in Fig. 6.1. This curve has an asymptote $n' = k_1'/a_1'$. It is interesting to make note of the outcome if $n' > k_1'/a_1'$. Under such a condition we only repeat the earlier situation where $n_o > k_1/a_1$. The values assigned to k_1 and a_1 by the super-debuggers are unsatisfactory, resulting in a further buildup of errors.

6.3 Economics with Penalty

To understand the economics involving penalty we can construct a hypothetical example. We shall consider a program of 10,000 machine language words having an initial error content equal to 1% of the total lines of code as observed by Shooman and Bolsky [6] in their experiment. Let us begin by assuming the following values for the parameters in the foregoing equations.

$n_o = 100$ (1% of the total number of words), $k_1 = 10$, $a_1 = 0.125$

$k_1' = 12.5$, $a_1' = 0.035$, $k_2 = 0.15$, $n_1 = 20$, $n_2 = 1$, $k_1'/a_1' = 357$

$k_1/a_1 = 80$

Reference [10] contains an analysis of the development costs of the Apollo Spacecraft guidance and control computer software. Complete statistics on the cost of software development were discussed in the report. The development cost per machine word ranges from \$60 on the lower limit to \$200 on the upper limit depending upon the complexity of the module. The development cost includes the cost of writing, debugging and testing. Debugging expense was rated at \$3000 per man-month.

We make a few more assumptions which are listed below.

*Number of men employed in Region 1 (i.e., $n > n_1$)	=	8
*Number of men employed in Region 2 (i.e., $n \leq n_1$)	=	4
Contract period	=	18 Months
Writing period	=	9 Months
Debugging and Testing	=	7.5 Months
Grace Period	=	1.5 Months
Debugging expense	=	\$3000/Man-Month
Development cost	=	\$ 50/Word
Profit	=	20% of development cost
Penalty per month of delay	=	1% of contract price

The development cost works out to be \$500,000. With 20% profit the contract price is \$600,000. Penalty is fixed at \$6000 for every month of delay.

In order to evaluate penalty it is essential to know the amount of time expended from the commencement of debugging. Using Eq. (5.7) we can write the effort (man-months) wasted in building the errors up to n' as

$$\tau' = \frac{1}{a_1} \ell n \left(\frac{n' - k_1/a_1}{n_o - k_1/a_1} \right) \tag{6.8}$$

Since the number of men employed are 8 and 4 in Regions 1 and 2, respectively, the amount of time (months) spent from the start of debugging is

$$T = \frac{\tau' + \tau_1}{8} + \frac{\tau_2 - \tau_1}{4} \quad ** \tag{6.9}$$

Let us consider an extreme case where n' = 350.

Using Eq. (6.8) we get $\tau' = 20.82$ man-months

From Eq. (6.2) we find $\tau_1 = 110.70$ man-months

and from Eq. (6.4) $\tau_2 = 136.70$ man-months

Using Eq. (6.9) $T = 22.94$ months

Since we have assumed a writing time of 9 months the total period = 22.94 + 9 = 31.94 months.

---

* For further details on Regions see Chapter 5.

**τ_2 is the number of man-months expended in reducing the errors from n' to n_2. Hence τ_1 is a subset of τ_2.

Penalty period = Total period - Contract period

= 31.94 - 18 = 13.94 months

Extensive debugging cost = \$3000/man-month x 136.70 man-months

= \$410,100 [using Eq. (6.5)]

Penalty in Dollars = \$6000 x 13.94 = \$83,640

Further debugging therefore costs \$410,100 + \$83,640 = \$493,740.

On the other hand, upon finding n' = 350 if the decision had been to rewrite then the total time spent from the beginning of the contract would be ' first writing period + $\tau'/8$ + redevelopment period.

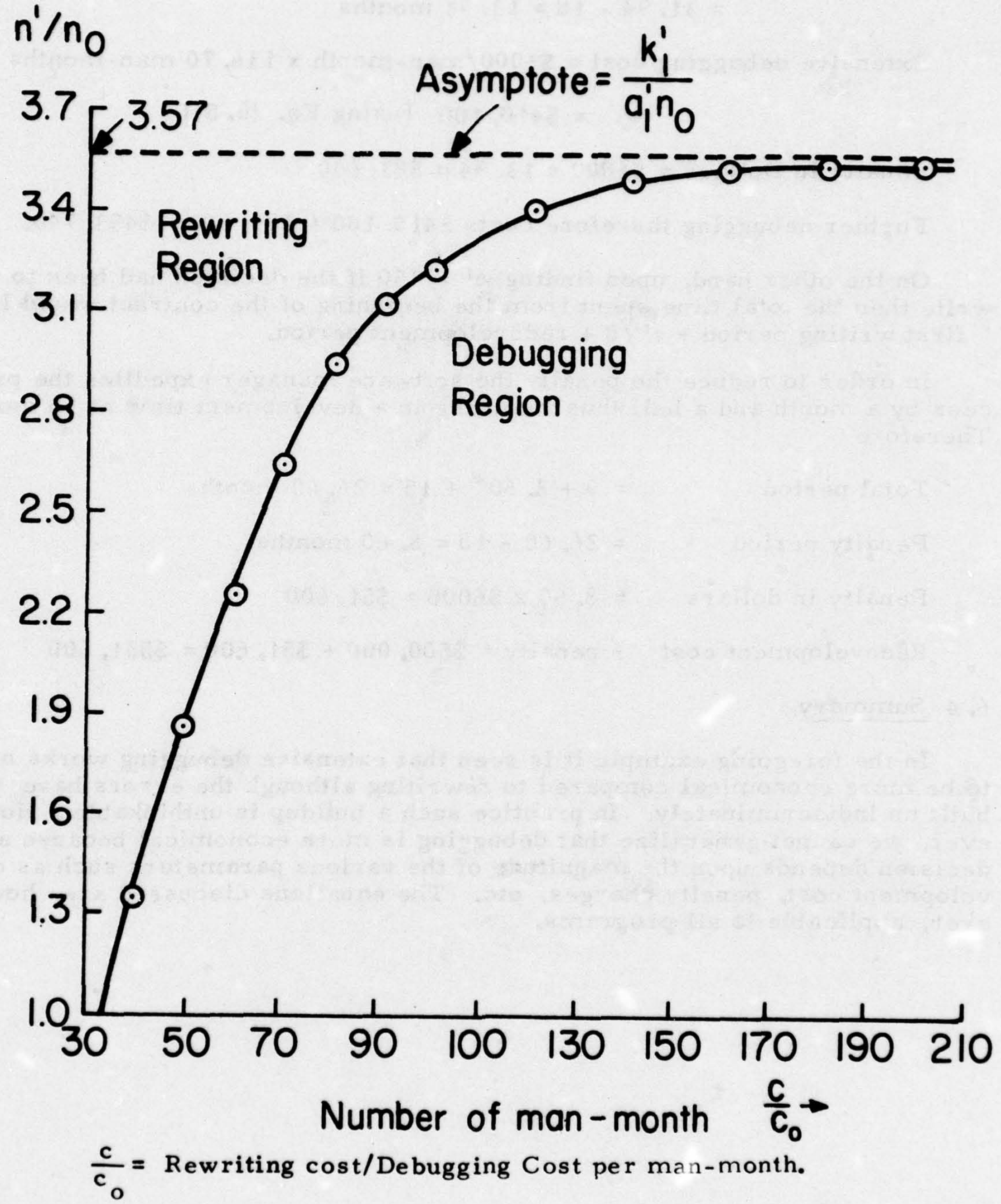
In order to reduce the penalty the software manager expedites the process by a month and a half thus resulting in a development time of 15 months. Therefore

Total period	= 9 + 2.60* + 15 = 26.60 months
Penalty period	= 26.60 - 18 = 8.60 months
Penalty in dollars	= 8.60 x \$6000 = \$51,600
Redevelopment cost	+ penalty = \$500,000 + \$51,600 = \$551,600

6.4 Summary

In the foregoing example it is seen that extensive debugging works out to be more economical compared to rewriting although the errors have built up indiscriminately. In practice such a buildup is unthinkable. However, we cannot generalize that debugging is more economical because a decision depends upon the magnitude of the various parameters such as development cost, penalty charges, etc. The equations discussed are, however, applicable to all programs.

*$\tau'/8 = 2.60$.



$\frac{c}{c_o}$ = Rewriting cost/Debugging Cost per man-month.

Fig. 6.1 Demarcation between debugging and rewriting regions (see Eq. (6.7))

CHAPTER 7

CONCLUSION

7.1 Summary

The development of an error model which accounted for error growth progressed through several stages, and culminated in the complex model developed in Chapter 5. At this point we are unable to justify the validity of those models because relevant experimental data is unavailable. However, basic assumptions and overall form of the results seem realistic. The model development led to the analysis of three cases. Assignment of insufficiently experienced debuggers results in a situation as depicted in Case 1 while a contrast to this is observed (Case 2) if superior debuggers are assigned to the same job. Very often programmers work on more than one project at a time and once the errors in one project are well under control the superior debuggers are assigned a new task which may subsequently result in an error buildup, and this is explained by Case 3.

7.2 Suggestions for Further Work

Having evolved reasonable error models, the next step is to evaluate the various parameters of the models, and test the validity of the model as a prediction test. In order to reduce the mathematical complexity, the change of manpower in all the models has been assumed to be abrupt. In practice the change is gradual.

Moreover the growth-decay parameters (a's and k's) are complex functions of the debugging time τ, manpower variation and rate of change of errors in the program. In order to formulate these functions it is necessary to study the changing manpower and the rate of error removal at every instant of debugging time in many projects. Another method of developing these functions is to measure the above parameters of individual programmers. Once these are available for every programmer, the parameters for the team could be established by including a mutual interaction coefficient (a factor due to coordination problems between programmers). Once these functions for the team are available, the error equations discussed in Chapter 5 could be applied to obtain a practical estimate of debugging effort required for any project.

APPENDIX I

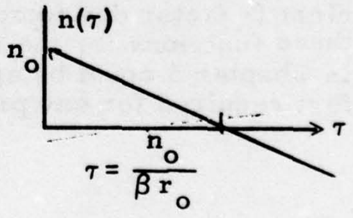
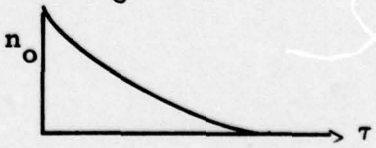
A summary of the basic models and assumptions of Chapters 4 and 5 appear in Table A-1 below.

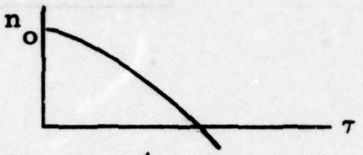
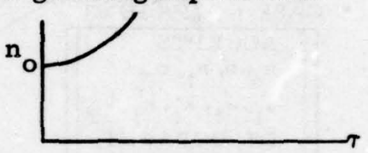
TABLE A-1 (Summary of Error Models)

BASIS EQUATION Eq. (4.3)

$$\frac{dn(\tau)}{d\tau} = r_g(\tau) - r_c(\tau)$$

$$n(\tau = 0) = n_o$$

CASE	EQUATION	SOLUTION
1. Generation and Correction Proportional to Detection		
$r_g(\tau) = \alpha\beta r_d(\tau)$	$\dot{n}(\tau) = \beta(\alpha-1) r_d(\tau)$	If $\alpha < 1$ $n(\tau)$ is a decreasing function
$r_c(\tau) = \beta r_d(\tau)$		If $\alpha > 1$ $n(\tau)$ is an increasing function
2. No generation $r_g(\tau) = 0$ $r_c(\tau) = \beta r_d(\tau)$ [Correction ~ Detection Detection = Constant]	$\dot{n}(\tau) = -\beta r_o$	$n(\tau) = n_o - (\beta r_o)\tau$ 
3. No generation $r_g(\tau) = 0$ $r_c(\tau) = \beta r_d(\tau)$ $r_d(\tau) = K_1 n(\tau)$ [Correction ~ Detection Detection ~ Error Present]	$\dot{n}(\tau) = -\beta K_1 n(\tau)$	$n(\tau) = n_o e^{-\beta K_1 \tau}$ 

CASE	EQUATION	SOLUTION
4. Generation Proportional to Product of Errors Present and Detection $r_g(\tau) = a\, r_d(\tau)\, n(\tau)$ $r_c(\tau) = b\, r_d(\tau)$ **[Correction Proportional to Detection]**	$\dot{n}(\tau) = [an(\tau)-b]r_d(\tau)$ For $r_d(\tau)$ = const. = r_o $\dot{n}(\tau) = [an(\tau)-b]r_o$	$n(\tau) = (n_o - b/a)e^{a r_o \tau} + b/a$ If $n_o < b/a$ then $n(\tau)$ is an inverted exponential  If $n_o > b/a$ then $n(\tau)$ is a growing exponential 

5. Generation Proportional to Number of Errors Present and Correction is either Manpower or Detection limited.
$r_g(\tau) = a_1\, n(\tau)$
$r_c(\tau) = \begin{cases} k_1 \text{ for } n(\tau) > n_1 \\ k_2 n(\tau) \text{ for } n(\tau) \leq n_1 \end{cases}$
	$\dot{n}(\tau) = -k_1 + a_1 n(\tau)$
	for $n(\tau) > n_1$
	$\dot{n}(\tau) = -k_2 n(\tau) + a_1 n(\tau)$
	for $n(\tau) \leq n_1$
		$\underline{n(\tau) > n_1}$
		$n(\tau) = (n_o - k_1/a_1)e^{a_1\tau} + k_1/a_1$
		(same form as Case 4)
		$\underline{n(\tau) \leq n_1}$
		$n(\tau) = n_1 \exp[(a_1 - k_2)(\tau - \tau_1)]$
		(i) If $1 < k_2/a_1$, then $n(\tau)$ decays exponentially as shown in Fig. 5.1b
		(ii) If $1 > k_2/a_1$, then $n(\tau)$ oscillates as shown in Fig. 5.1c.

APPENDIX II

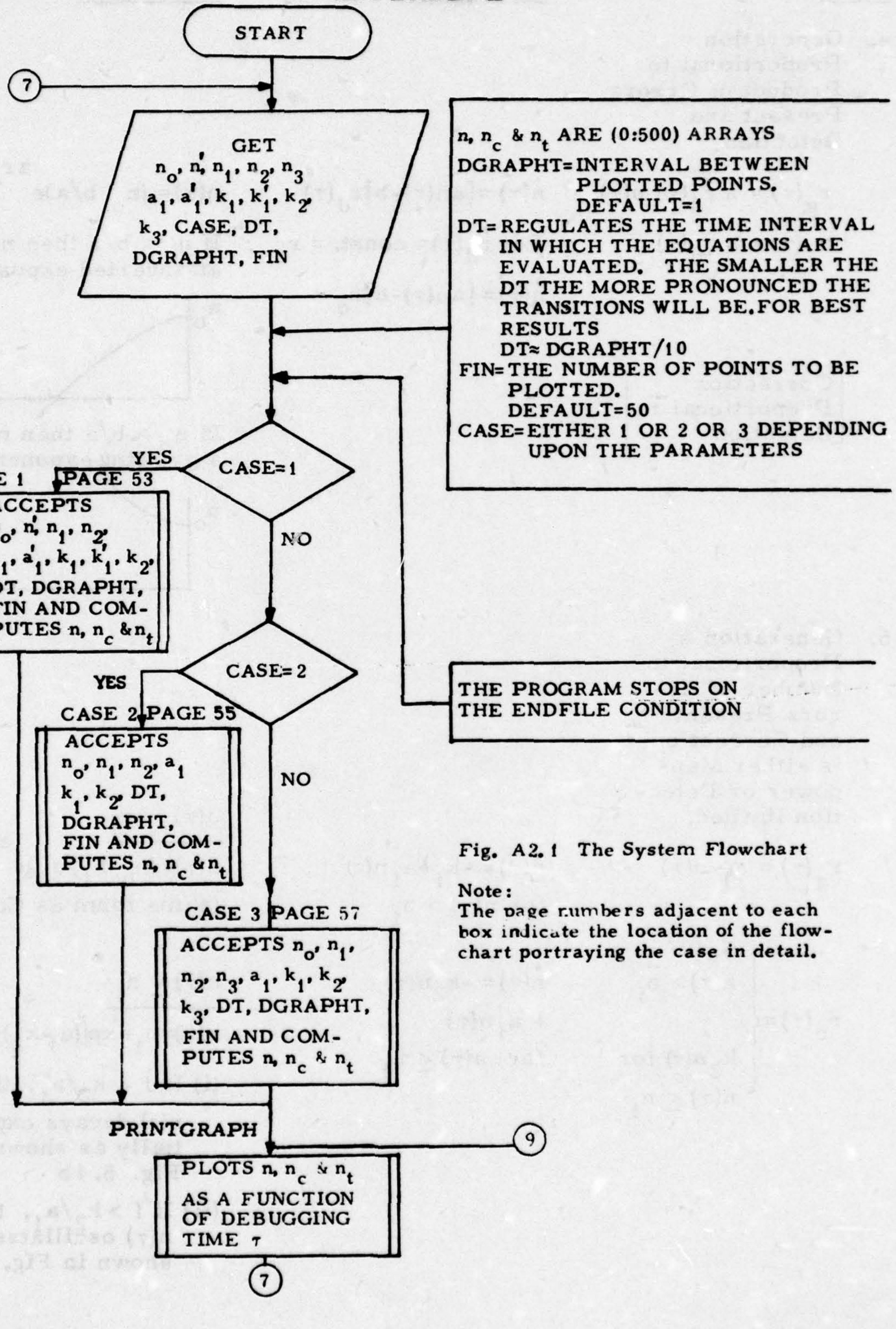

START
7
GET
n_o, n_o', n_1, n_2, n_3
$a_1, a_1', k_1, k_1', k_2,$
k_3, CASE, DT,
DGRAPHT, FIN
n, n_c & n_t ARE (0:500) ARRAYS
DGRAPHT=INTERVAL BETWEEN PLOTTED POINTS. DEFAULT=1
DT=REGULATES THE TIME INTERVAL IN WHICH THE EQUATIONS ARE EVALUATED. THE SMALLER THE DT THE MORE PRONOUNCED THE TRANSITIONS WILL BE.FOR BEST RESULTS DT≈ DGRAPHT/10
FIN=THE NUMBER OF POINTS TO BE PLOTTED. DEFAULT=50
CASE=EITHER 1 OR 2 OR 3 DEPENDING UPON THE PARAMETERS
CASE=1
YES
NO
CASE 1
PAGE 53
ACCEPTS $n_o, n_o', n_1, n_2,$ $a_1, a_1', k_1, k_1', k_2,$ DT, DGRAPHT, FIN AND COMPUTES n, n_c &n_t
CASE= 2
YES
NO
CASE 2
PAGE 55
ACCEPTS n_o, n_1, n_2, a_1 k_1, k_2, DT, DGRAPHT, FIN AND COMPUTES n, n_c &n_t
THE PROGRAM STOPS ON THE ENDFILE CONDITION
CASE 3
PAGE 57
ACCEPTS $n_o, n_1,$ $n_2, n_3, a_1, k_1, k_2,$ k_3, DT, DGRAPHT, FIN AND COMPUTES n, n_c & n_t
PRINTGRAPH
9
PLOTS n, n_c & n_t AS A FUNCTION OF DEBUGGING TIME τ
7

Fig. A2.1 The System Flowchart

Note:
The page numbers adjacent to each box indicate the location of the flowchart portraying the case in detail.

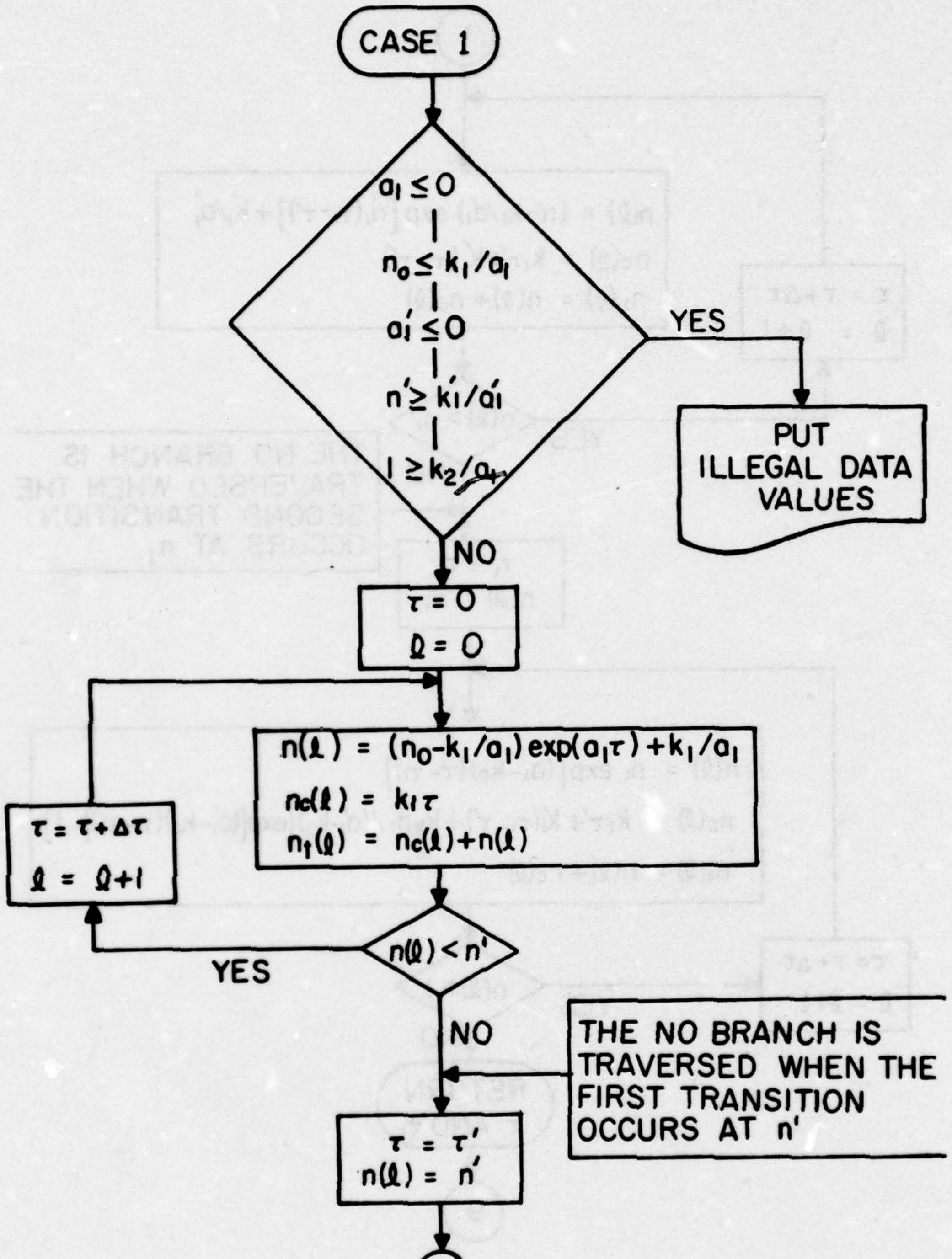


Fig. A2. 2 Part 1 of the flowchart for CASE 1

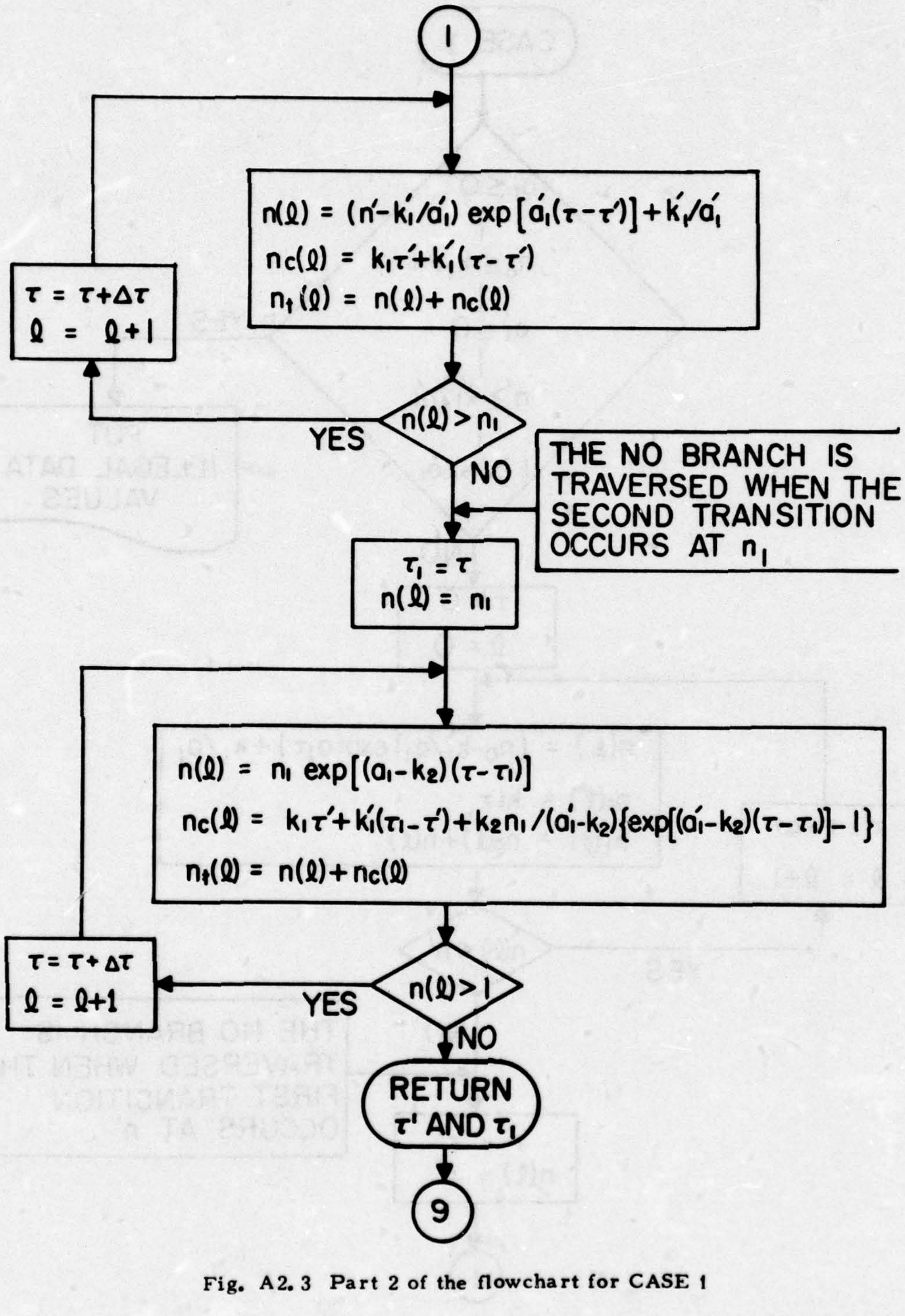

1
n(ℓ) = (n′ − k′₁/a′₁) exp[a′₁(τ − τ′)] + k′₁/a′₁
nc(ℓ) = k₁τ′ + k′₁(τ − τ′)
nₜ(ℓ) = n(ℓ) + nc(ℓ)
τ = τ + Δτ
ℓ = ℓ + 1
n(ℓ) > n₁
YES
NO
THE NO BRANCH IS TRAVERSED WHEN THE SECOND TRANSITION OCCURS AT n₁
τ₁ = τ
n(ℓ) = n₁
n(ℓ) = n₁ exp[(a₁ − k₂)(τ − τ₁)]
nc(ℓ) = k₁τ′ + k′₁(τ₁ − τ′) + k₂n₁/(a′₁ − k₂){exp[(a′₁ − k₂)(τ − τ₁)] − 1}
nₜ(ℓ) = n(ℓ) + nc(ℓ)
τ = τ + Δτ
ℓ = ℓ + 1
n(ℓ) > 1
YES
NO
RETURN τ′ AND τ₁
9

Fig. A2.3 Part 2 of the flowchart for CASE 1

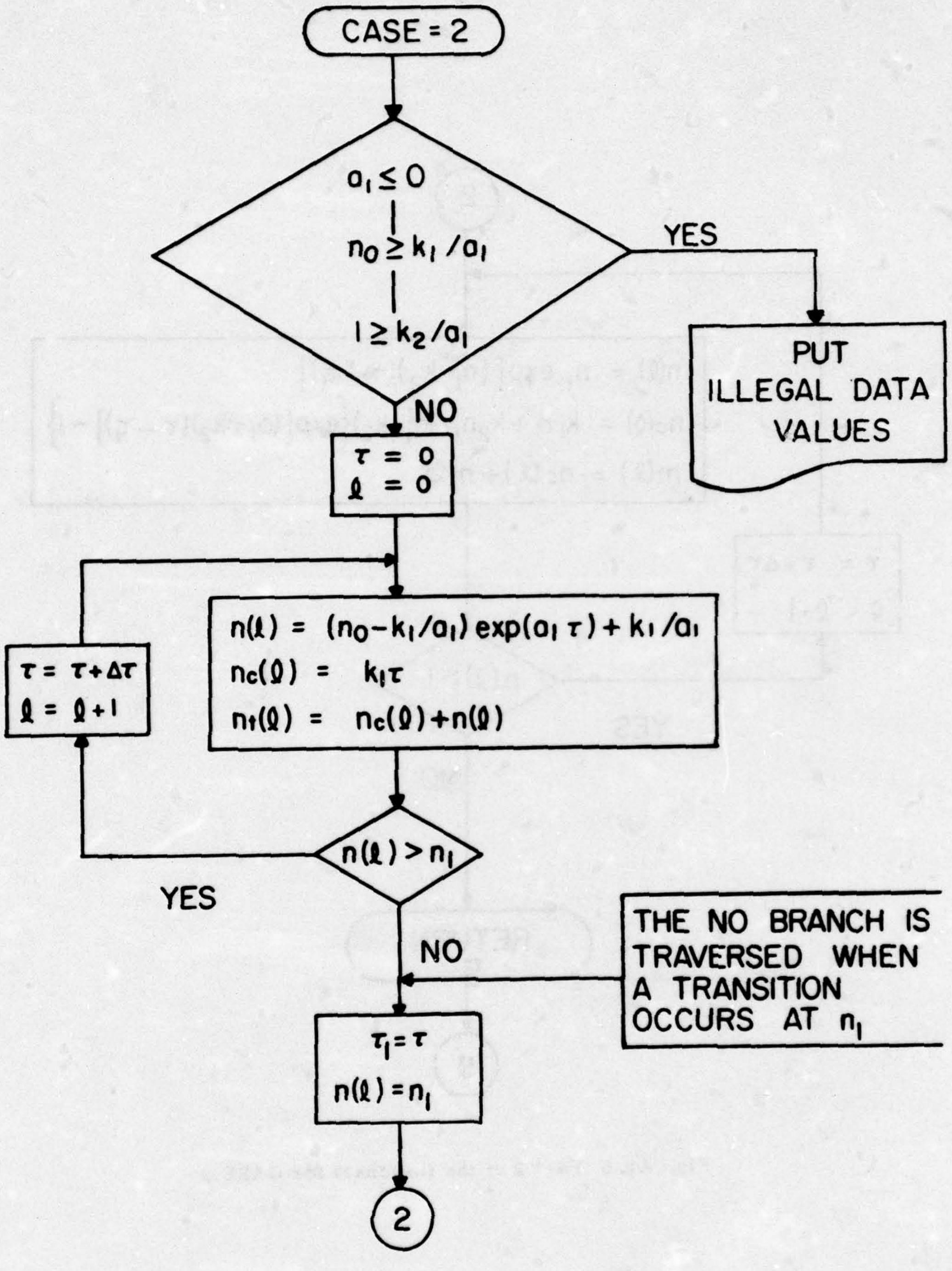


Fig. A2.4 Part 1 of the flowchart for CASE 2

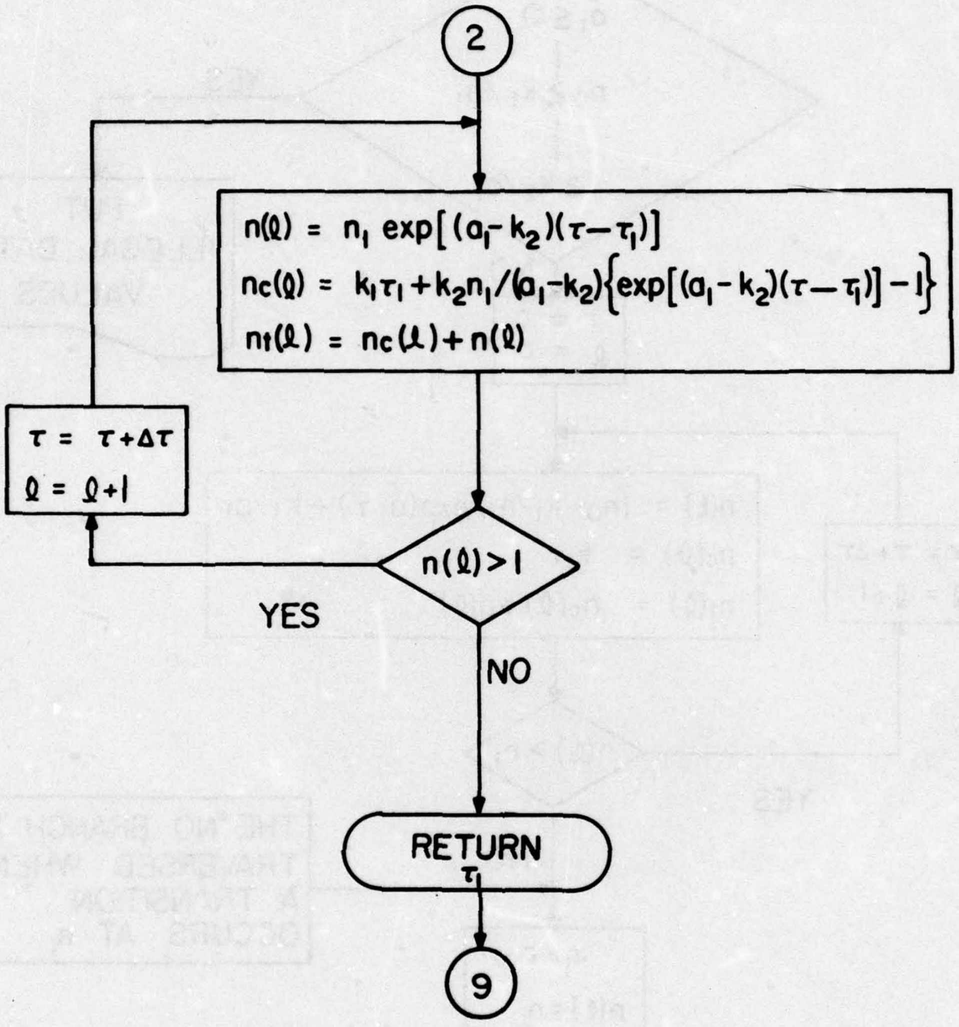

2
$n(\ell) = n_1 \exp[(a_1 - k_2)(\tau - \tau_1)]$
$nc(\ell) = k_1\tau_1 + k_2 n_1/(a_1 - k_2)\{\exp[(a_1 - k_2)(\tau - \tau_1)] - 1\}$
$nt(\ell) = nc(\ell) + n(\ell)$
$\tau = \tau + \Delta\tau$
$\ell = \ell + 1$
$n(\ell) > 1$
YES
NO
RETURN τ_1
9

Fig. A2. 5 Part 2 of the flowchart for CASE 2

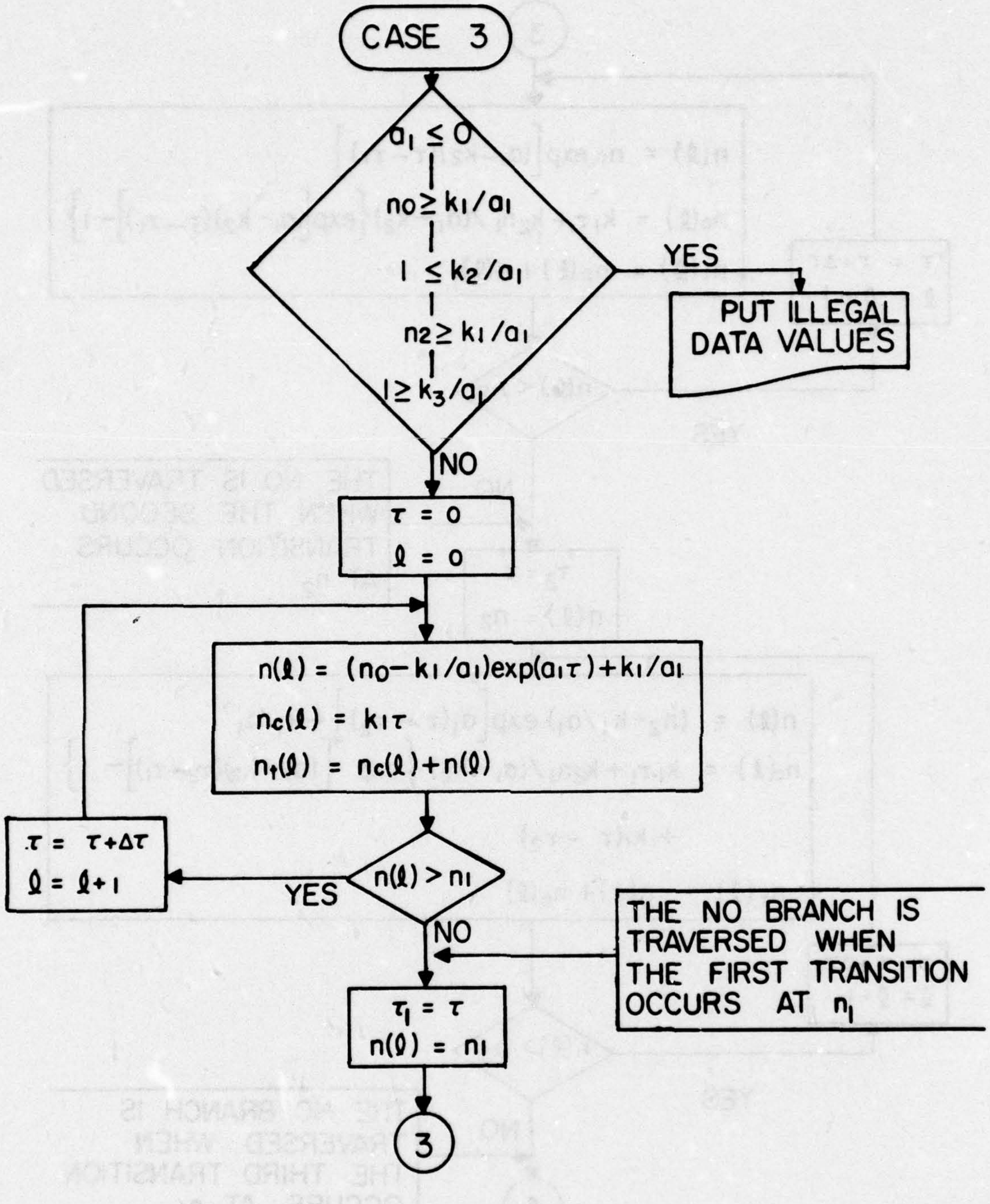


Fig. A2.6 Part 1 of the flowchart for CASE 3

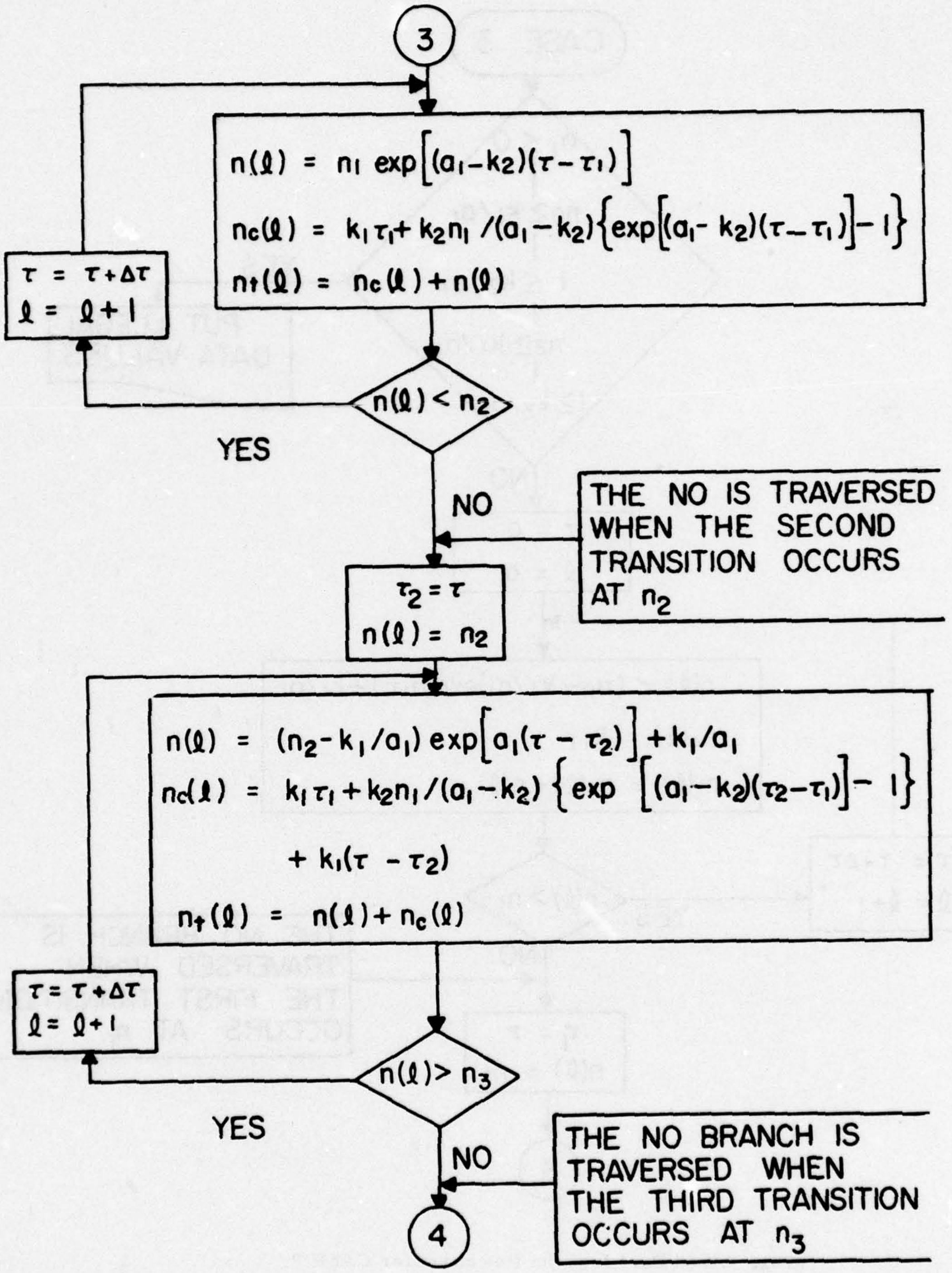

3
$n(\ell) = n_1 \exp[(a_1-k_2)(\tau-\tau_1)]$
$n_c(\ell) = k_1\tau_1 + k_2 n_1/(a_1-k_2)\{\exp[(a_1-k_2)(\tau-\tau_1)]-1\}$
$n_t(\ell) = n_c(\ell)+n(\ell)$
$\tau = \tau+\Delta\tau$
$\ell = \ell+1$
$n(\ell) < n_2$
YES
NO
THE NO IS TRAVERSED WHEN THE SECOND TRANSITION OCCURS AT n_2
$\tau_2 = \tau$
$n(\ell) = n_2$
$n(\ell) = (n_2-k_1/a_1)\exp[a_1(\tau-\tau_2)]+k_1/a_1$
$n_c(\ell) = k_1\tau_1 + k_2 n_1/(a_1-k_2)\{\exp[(a_1-k_2)(\tau_2-\tau_1)]-1\} + k_1(\tau-\tau_2)$
$n_t(\ell) = n(\ell)+n_c(\ell)$
$\tau = \tau+\Delta\tau$
$\ell = \ell+1$
$n(\ell) > n_3$
YES
NO
THE NO BRANCH IS TRAVERSED WHEN THE THIRD TRANSITION OCCURS AT n_3
4

Fig. A2.7 Part 2 of the flowchart for CASE 3

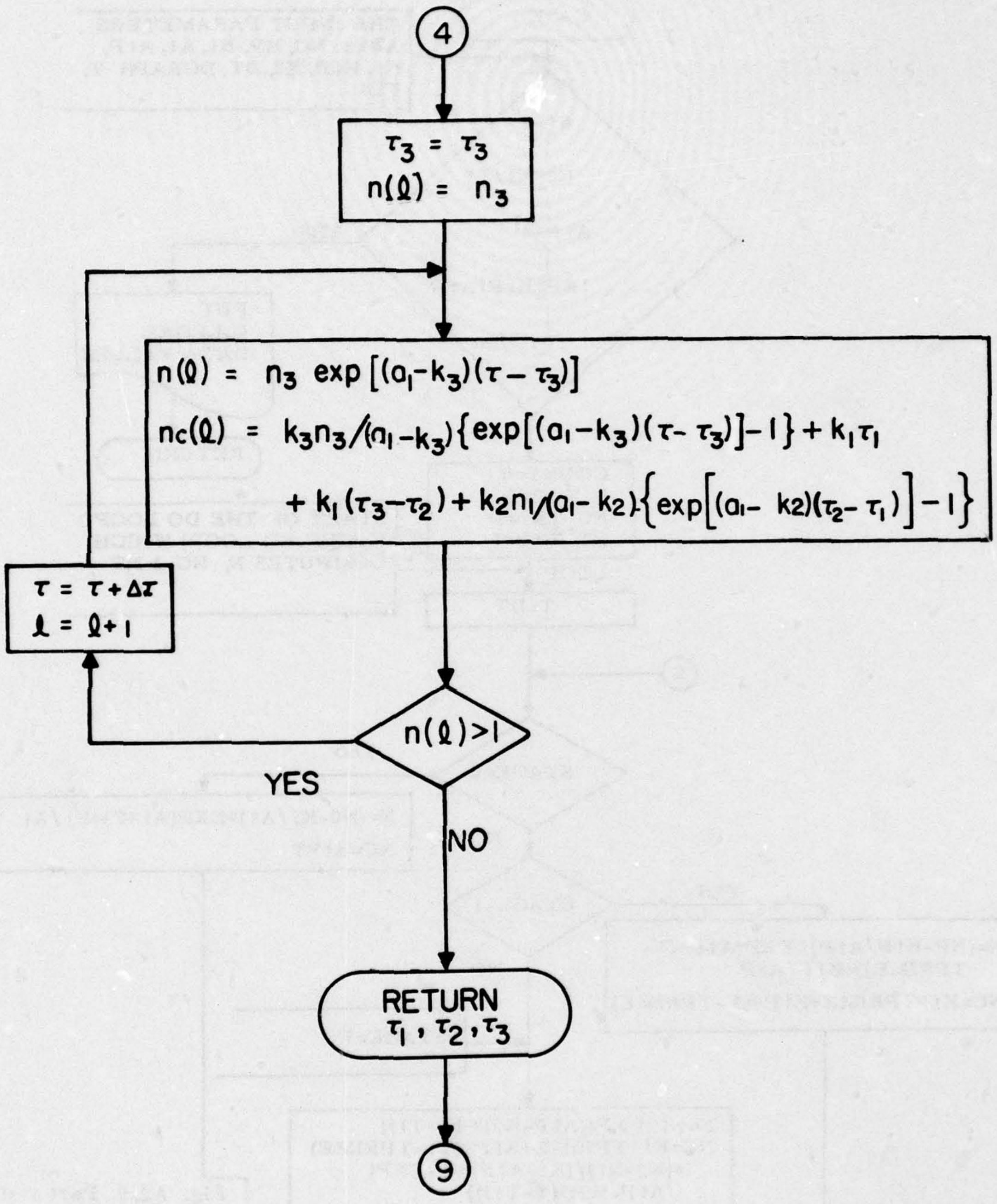

4
$\tau_3 = \tau_3$
$n(\ell) = n_3$
$n(\ell) = n_3 \exp[(a_1-k_3)(\tau-\tau_3)]$
$nc(\ell) = k_3 n_3/(a_1-k_3)\{\exp[(a_1-k_3)(\tau-\tau_3)]-1\}+k_1\tau_1$
$+k_1(\tau_3-\tau_2)+k_2 n_1/(a_1-k_2)\{\exp[(a_1-k_2)(\tau_2-\tau_1)]-1\}$
$\tau = \tau+\Delta\tau$
$\ell = \ell+1$
$n(\ell)>1$
YES
NO
RETURN
τ_1, τ_2, τ_3
9

Fig. A2.8 Part 3 of the flowchart for CASE 3

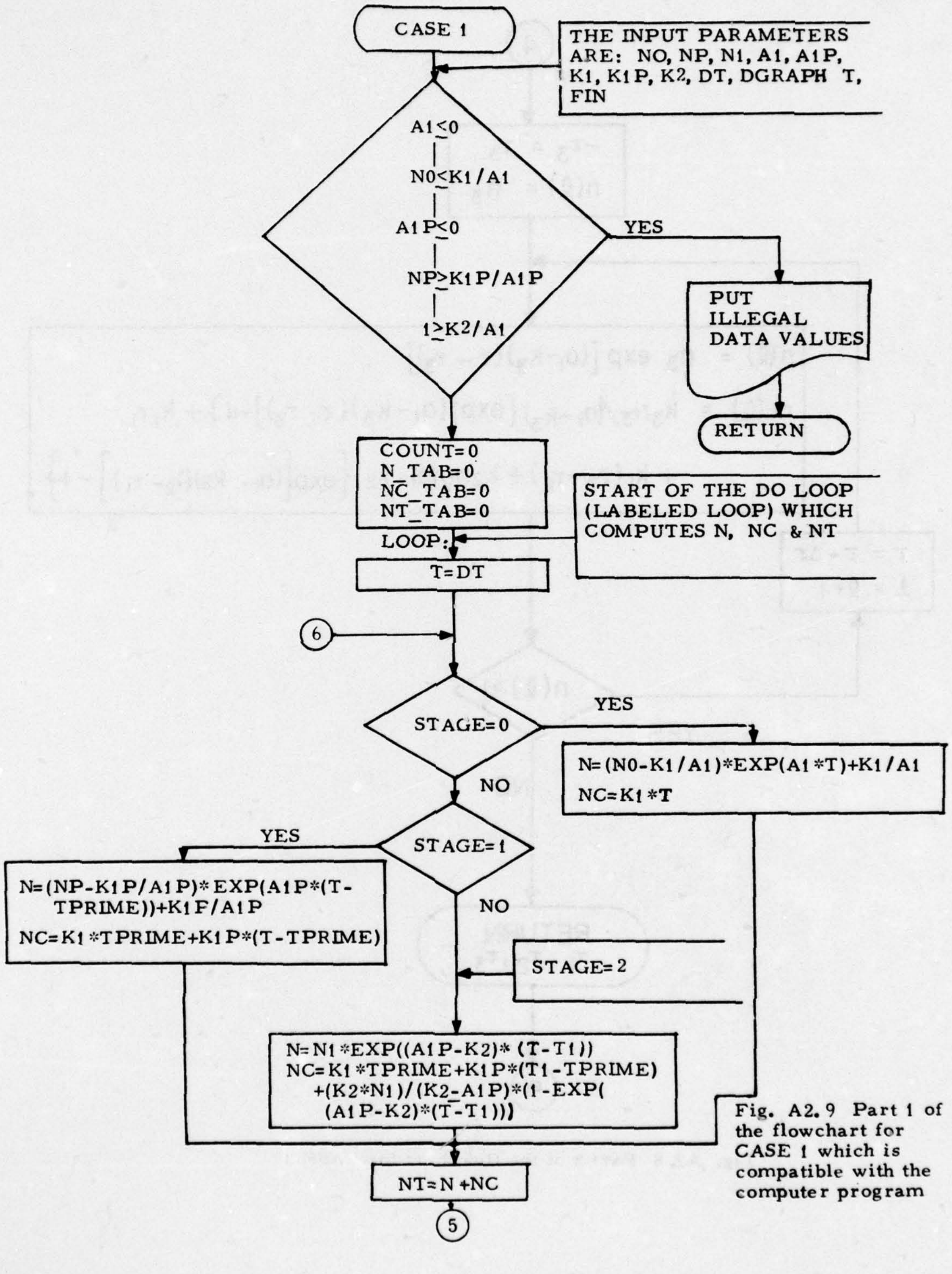

CASE 1
THE INPUT PARAMETERS ARE: NO, NP, N1, A1, A1P, K1, K1P, K2, DT, DGRAPH T, FIN
A1≤0
N0<K1/A1
A1P≤0
NP≥K1P/A1P
1≥K2/A1
YES
PUT ILLEGAL DATA VALUES
RETURN
COUNT=0
N_TAB=0
NC_TAB=0
NT_TAB=0
LOOP:
START OF THE DO LOOP (LABELED LOOP) WHICH COMPUTES N, NC & NT
T=DT
6
STAGE=0
YES
N=(N0-K1/A1)*EXP(A1*T)+K1/A1
NC=K1*T
NO
STAGE=1
YES
N=(NP-K1P/A1P)*EXP(A1P*(T-TPRIME))+K1F/A1P
NC=K1*TPRIME+K1P*(T-TPRIME)
NO
STAGE=2
N=N1*EXP((A1P-K2)*(T-T1))
NC=K1*TPRIME+K1P*(T1-TPRIME)
+(K2*N1)/(K2-A1P)*(1-EXP(
(A1P-K2)*(T-T1)))
NT=N+NC
5

Fig. A2.9 Part 1 of the flowchart for CASE 1 which is compatible with the computer program

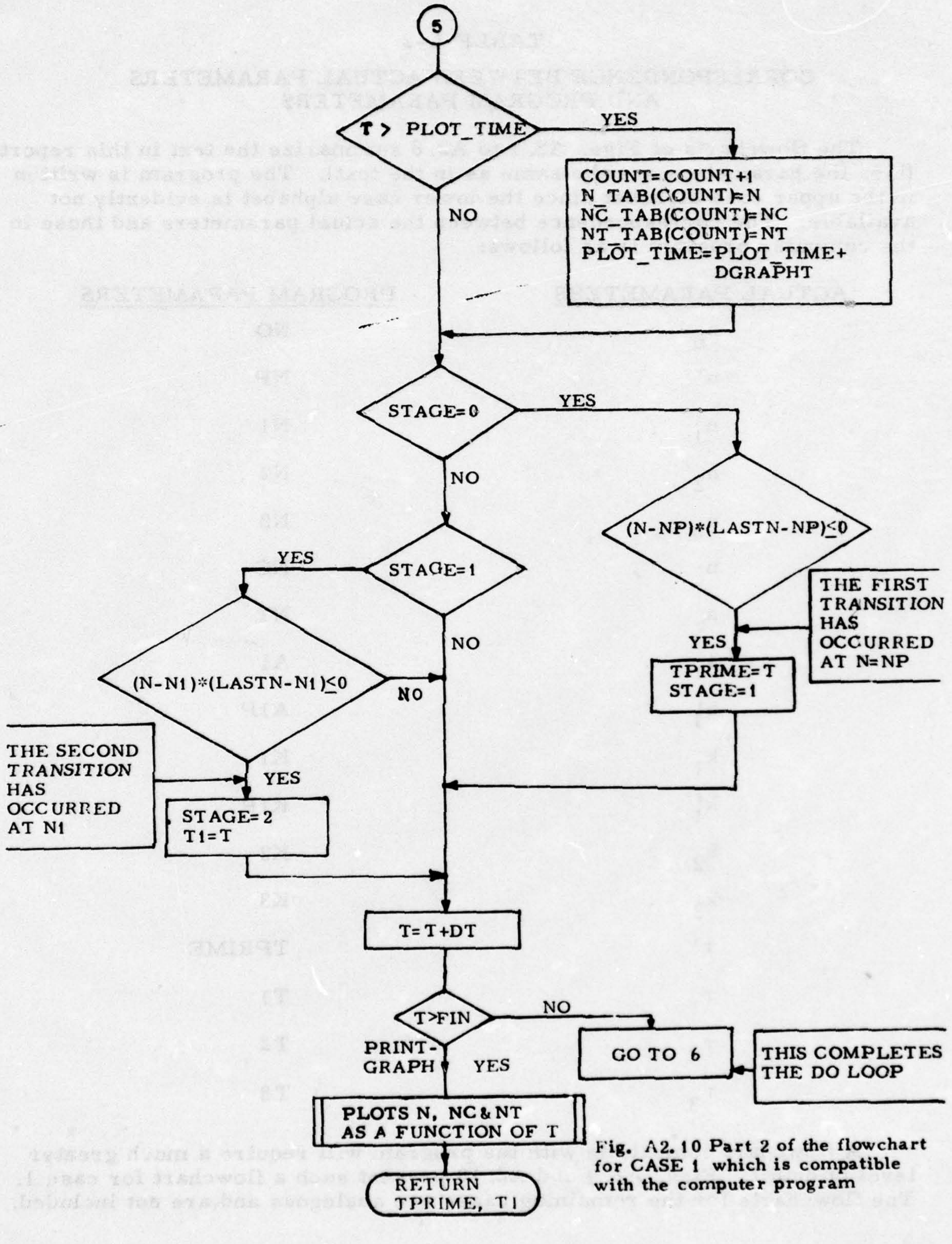


Fig. A2. 10 Part 2 of the flowchart for CASE 1 which is compatible with the computer program

TABLE A-2

CORRESPONDENCE BETWEEN ACTUAL PARAMETERS AND PROGRAM PARAMETERS

The flowcharts of Figs. A2. 1 to A2. 8 summarize the text in this report (i. e. the parameters are the same as in the text). The program is written in the upper case alphabet since the lower case alphabet is evidently not available. The correspondence between the actual parameters and those in the computer program is as follows:

ACTUAL PARAMETERS	PROGRAM PARAMETERS
n_o	NO
n'	NP
n_1	N1
n_2	N2
n_3	N3
n_c	NC
n_t	NT
a_1	A1
a'_1	A1P
k_1	K1
k'_1	K1P
k_2	K2
k_3	K3
τ'	TPRIME
τ_1	T1
τ_2	T2
τ_3	T3

A flowchart compatible with the program will require a much greater level of detail. Figs. A2. 9 and A2. 10 present such a flowchart for case 1. The flowcharts for the remaining cases are analogous and are not included.

```
 (STRINGRANGE,SUBSCRIPTRANGE):
  MODELS: PROCEDURE OPTIONS (MAIN);
/*        EXPLANATIONS OF DATA VARIABLES

  INPUT VARIABLES

  CASE1                             THE CORRESPONDENCE BETWEEN
  NO,NP,N1,A1,A1P,K1,               THESE VARIABLES AND THOSE
  K1P,K2,DT,DGRAPHT,FIN             IN THE TEXT (WHICH ARE
                                    WRITTEN IN LOWER CASE)
                                    FOR ALL CASES IS SHOWN IN
  CASE2                             TABLE A-2 ON PAGE 62.
  NO,N1,A1,K1,K2

  CASE3
  NO,N1,N2,N3,A1,K1,K2,K3

  DGRAPHT                           INTERVAL BETWEEN PLOTTED
                                    POINTS.  DEFAULT = 1

  DT                                REGULATES THE TIME INTERVAL
                                    IN WHICH THE EQUATIONS ARE
                                    EVALUATED.  THE SMALLER THE
                                    VALUE OF DT THE CLOSER THE
                                    TRANSITIONS WILL BE.  FOR
                                    BEST RESULTS DT=DGRAPHT/10.

  FIN                               THE NUMBER OF POINTS TO BE
                                    PLOTTED.  DEFAULT = 50.

  ********** EXAMPLES OF DATA CARDS **********************

  CASE=2, NO=100, K1=10, K2=.2, N1=25, A1=.0665  ;

  CASE=3, NO=100, K1=10, A1=.026, N1=30, K2=.01, N2=40,
          N3=20, K3=.15    ;

  *****************************************************  */
```

```
DECLARE (NO, NP, N1, A1, A1P, K1, K1P, K2, K3, N2, N3)
        FLOAT INITIAL (0),
        (N_TAB, NC_TAB, NT_TAB ) (0:500) FLOAT( 16 ),
        (T, TPRIME, T1, T2, T3, N, NC, NT) FLOAT (16)
        INITIAL (0E0);
DECLARE ( COUNT, LASTN, DGRAPHT, DT , FIN, PLOT_TIME)
        FLOAT, I FIXED BIN (31),
        STAGE FIXED BINARY (31) INITIAL (1);

/* PLOT_TIME IS THE INSTANT OF TIME AT WHICH THE FUNCTION
   IS PLOTTED, WHEREAS T IS THE INSTANT OF TIME AT WHICH
   THE EQUATIONS ARE COMPUTED.  THE TIME INTERVAL IN WHICH
   THE EQUATIONS ARE COMPUTED IS MUCH SMALLER THAN THE
   INTERVAL BETWEEN THE PLOTTED POINTS ( TEN POINTS ARE
   COMPUTED, AND ONLY ONE PLOTTED).  T<< PLOT_TIME IS
   DESIRED BECAUSE A CONTINUOUS CHANGE OF THE VARIABLES
   N, NC AND NT IS AVAILABLE (IN THE ASSUMED INPUT DATA).
   A SMALL T IS EQUIVALENT TO A DAY-TO-DAY PROGRESS IN A
   REAL-LIFE SITUATION, WHILE A LARGE PLOT_TIME IS EQUIVALENT
   TO A CO-ORDINATE IN A PERFORMANCE CHART OVER A LONG
   PERIOD, SAY ONE YEAR.  THE RECORDING OF THE RESULTS,
   HOWEVER, MAY TAKE PLACE EVERY 15 DAYS.              */
```

```
ON ENDFILE (SYSIN) BEGIN;
PUT PAGE EDIT ('END OF DATA ENCOUNTERED. ', ' IF ANY '||
'DATA SETS SEEM TO HAVE BEEN SKIPPED, CHECK TO SEE IF YOU'
|| ' PUT IN THE SEMICOLON', ' ******TERMINATING RUN ****
') (COL(1),A);
STOP;
END;

ON NAME (SYSIN) BEGIN;
PUT PAGE EDIT (' ERROR ***** ONE OF THE INPUT DATA ITEMS '
|| 'IS NOT AS EXPECTED. THE ONLY LEGAL DATA ITEMS ARE :',
' NO, NP, N1, N2, N3, A1, A1P, K1, K1P, K2, K3, DT, FIN, '
||'DGRAPHT',
' THE LAST THREE BEING OPTIONAL', 'PROBABLE CAUSE IS' ||
' KEYPUNCH ERROR IN PUNCHING THE DATA', ' THE PART WHICH '
||'CAUSED THE TROUBLE IS:'||DATAFIELD,
' ***** TERMINATING RUN *****') ( A,SKIP);
STOP;
END;

ON ERROR SNAP BEGIN;
ON ERROR SYSTEM;
PUT SKIP (4) LIST (' AN INTERNAL PROGRAM ERROR HAS '||
'OCCURED. PLEASE SHOW THIS PRINTOUT TO S. NATARAJAN');
PUT SKIP LIST ('ONCODE=', ONCODE);
PUT SKIP (4) DATA;
STOP;
END;

/* INITIALIZATIONS    */
T=0;  FIN=50 ;  DT=1E-1;  DGRAPHT=1;
DO WHILE ('1'B);
GET DATA (NO,NP,N1,N2,N3, A1,A1P,K1,K1P,K2,K3,DT,DGRAPHT,
CASE,FIN);
STAGE=0;
/* STAGE=0 INDICATES WE ARE BEFORE THE FIRST TRANSITION
STAGE=1 INDICATES AFTER FIRST TRANSITION
STAGE=2 INDICATES AFTER SECOND TRANSITION                  */
PLOT_TIME=DGRAPHT;
N_TAB(0)=NO;
NC_TAB(0)=0 ;
NT_TAB(0)=N_TAB(0);
LASTN=N_TAB(0);

/* LASTN IS THE PREVIOUS VALUE OF N    */
IF CASE=1 THEN CALL CASE1 ;
          ELSE IF CASE=2 THEN CALL CASE2;
                         ELSE CALL CASE3;
END;
```

```
CASE1: PROC;
COUNT=0E0;
N_TAB=0E0;  NC_TAB=0E0;  NT_TAB=0E0;
PUT PAGE EDIT ('CASE 1')(COL(44),A);
PUT SKIP (3);
PUT EDIT ('NO =',NO,'NP =',NP,'N1 =',N1)
        (COL(14),A,COL(19),F(3),COL(38),A,COL(43),F(3),
        COL(63),A,COL(68),F(2))
         ('A1 =',A1,'A1P =',A1P,'K1 =',K1)
         (COL(14),A,COL(19),F(5,3),COL(38),A,COL(44),F(6,4),
        COL(63),A,COL(68),F(2))
         ('K1P =',K1P,'K2 =',K2,'DT =',DT)
         (COL(14),A,COL(20),F(4,1),COL(38),A,COL(43),F(5,3),
        COL(63),A,COL(68),F(5,3))
         ('DGRAPHT =',DGRAPHT,'FIN =',FIN)
         (COL(14),A,COL(24),F(1),COL(38),A,COL(44),F(2));
PUT SKIP (6);
IF (A1<=0) | (NO<=K1/A1) | (A1P=0) | (NP>=K1P/A1P)
THEN GO TO ERROR      ;
GO TO RESUME;
ERROR:  PUT SKIP LIST ('ILLEGAL DATA VALUES, SKIPPING SET');
RETURN;

RESUME:
LOOP: DO T= DT TO FIN BY DT;
/* COMPUTE THE EQUATIONS */
IF STAGE =0 THEN DO;
    N=(NO-K1/A1)* EXP(A1*T)+ K1/A1 ;
    NC= K1*T;
    END;
ELSE IF STAGE=1 THEN DO;
    N=(NP-K1P/A1P) * EXP(A1P*(T-TPRIME)) + K1P/A1P;
    NC=K1*TPRIME + K1P*(T-TPRIME);
    END;
ELSE DO;
    N=N1*EXP((A1P-K2)*(T-T1)) ;
    NC=K1*TPRIME + K1P*(T1-TPRIME) + (K2*N1)/(K2-A1P)*
    (1-EXP((A1P-K2)*(T-T1 ))) ;
    END;
```

```
/* IN ANY EVENT    */  NT=N + NC;
IF T>PLOT_TIME THEN DO;    /* IT'S TIME TO RECORD ANOTHER
                              SET OF POINTS FOR LATER
                              GRAPHING                  */
COUNT=COUNT + 1;
N_TAB(COUNT) = N;
NC_TAB(COUNT) = NC;
NT_TAB(COUNT) = NT;
PLOT_TIME = PLOT_TIME + DGRAPHT;
END;
IF STAGE = 0 THEN IF (N-NP)*(LASTN-NP)<=0E0 THEN DO;
/* THE ERRORS HAVE INCREASED TO N' WHICH CORRESPONDS TO
   THE FIRST TRANSITION IN THE ERROR EQUATION .  */
TPRIME=T;
STAGE = 1;
END;
                                            ELSE;
ELSE IF STAGE = 1 THEN IF (N-N1) *(LASTN-N1)<=0E0 THEN DO;
/* THE ERRORS HAVE DECREASED TO N1 WHICH CORRESPONDS TO
   THE SECOND TRANSITION IN THE ERROR EQUATION. */
STAGE = 2;
T1=T;
END;
LASTN=N;
END LOOP;
CALL PRINTGRAPH;
PUT SKIP DATA ( TPRIME, T1);
END CASE1;
```

```
CASE2:PROC;
COUNT=0;
N_TAB=0E0;  NC_TAB=0E0;  NT_TAB=0E0;
PUT PAGE EDIT ('CASE 2')(COL(44),A);
PUT SKIP(3);
PUT EDIT ('N0 =',N0,'N1 =',N1,'A1 =',A1)
        (COL(14),A,COL(19),F(3),COL(38),A,COL(43),F(2),
        COL(63),A,COL(68),F(6,4))
        ('K1 =',K1,'K2 =',K2,'DT =',DT)
        (COL(14),A,COL(19),F(2),COL(38),A,COL(43),F(5,3),
        COL(63),A,COL(68),F(5,3))
        ('FIN =',FIN,'DGRAPHT =',DGRAPHT)
        (COL(14),A,COL(20),F(2),COL(38),A,COL(48),F(1));
PUT SKIP (6);

LOOP: DO T= DT TO FIN BY DT;
/*  COMPUTE THE EQUATIONS */
IF STAGE = 0 THEN DO;
    N=(N0-K1/A1)*EXP(A1*T) + K1/A1;
    NC = K1*T;
    END;
ELSE DO;
    N=N1*EXP((A1-K2)*(T-T1));
    NC=K1*T1 + (K2*N1)/(K2-A1)*(1E0 - EXP((A1-K2)*(T - T1)));
    END;
NT=N + NC;
IF T>PLOT_TIME THEN DO;    /*  IT'S TIME TO RECORD ANOTHER
                               SET OF POINTS FOR LATER
                               GRAPHING                  */
  COUNT=COUNT+1;
  N_TAB(COUNT) =N;
  NC_TAB(COUNT)=NC;
  NT_TAB(COUNT)=NT;
  PLOT_TIME=PLOT_TIME+DGRAPHT;
  END;
IF STAGE=0 THEN IF (N-N1)*(LASTN-N1)<=0E0 THEN DO;
/* THE ERRORS HAVE DECREASED TO N1 WHICH CORRESPUNDS TO
   THE FIRST TRANSITION IN THE ERROR EQUATION.          */
    T1=T;
    STAGE=1;
    END;
LASTN=N;
END LOOP;
CALL PRINTGRAPH;
PUT SKIP DATA(T1);
END CASE2;
```

```
CASE3:PROC;
DECLARE ( NCT2,  /* =NC(T2)  */
          NCT3   /*=NC(T3)  */) FLOAT INIT(0E0);
COUNT=0;
N_TAB=0E0;  NC_TAB=0E0;  NT_TAB=0E0;
PUT PAGE EDIT ('CASE 3')(COL(44),A);
PUT SKIP (3);
PUT EDIT ('N0 =',N0,'N1 =',N1,'N2 =',N2)
          (COL(14),A,COL(19),F(3),COL(38),A,COL(43),F(2),
          COL(63),A,COL(68),F(2))
          ('N3 =',N3,'A1 =',A1,'K1 =',K1)
          (COL(14),A,COL(19),F(2),COL(38),A,COL(43),F(6,4),
          COL(63),A,COL(68),F(2))
          ('K2 =',K2,'K3 =',K3,'DT =',DT)
          (COL(14),A,COL(19),F(6,4),COL(38),A,COL(43),F(5,3),
          COL(63),A,COL(68),F(5,3))
          ('FIN =',FIN,'DGRAPHT =',DGRAPHT)
          (COL(14),A,COL(20),F(2),COL(38),A,COL(48),F(1));
PUT SKIP (6);

LOOP: DO T= DT TO FIN BY DT;
/* COMPUTE THE EQUATIONS                                  */
IF STAGE =0 THEN DO;
    N=(N0-K1/A1)* EXP(A1*T) + K1/A1;
    NC=K1*T;
    END;
ELSE IF STAGE = 1 THEN DO;
    N=N1*EXP((A1-K2)*(T-T1));
    NC= (K2*N1)/(A1-K2)*(EXP((A1-K2)*(T-T1))-1E0)+K1*T1;
    END;
ELSE IF STAGE=2 THEN DO;
    N=(N2-K1/A1)*EXP(A1*(T-T2))+K1/A1;
    NC=NCT2+K1*(T-T2);
    END;
```

```
ELSE IF STAGE=3 THEN DO;
    N=N3*EXP((A1-K3)*(T-T3));
    NC=(K3*N3)/(K3-A1)*(1-EXP((A1-K3)*(T-T3)))+NCT3;
    END;
NT=NC+N;
IF T>PLOT_TIME THEN DO;    /*  IT'S TIME TO RECORD ANOTHER
                               SET OF POINTS FOR LATER
                               GRAPHING                       */
  COUNT=COUNT+1;
  N_TAB(COUNT) =N;
  NC_TAB(COUNT)=NC;
  NT_TAB(COUNT)=NT;
  PLOT_TIME=PLOT_TIME+DGRAPHT;
  END;
IF STAGE=0 THEN IF (N-N1)*(LASTN-N1)<=0E0 THEN DO;
/* THE ERRORS HAVE DECREASED TO N1 WHICH CORRESPONDS TO
   THE FIRST TRANSITION IN THE ERROR EQUATION.         */
  T1=T;
  STAGE=1;
  END;
  ELSE;
ELSE IF STAGE=1 THEN IF (N-N2)*(LASTN-N2)<=0E0 THEN DO;
/* THE ERRORS HAVE UNFORTUNATELY INCREASED TO N2 WHICH
   CORRESPONDS TO THE SECOND TRANSITION IN THE ERROR
   EQUATION.                                           */
  T2=T;
  NCT2=NC;
  STAGE=2;
  END;
  ELSE;
ELSE IF STAGE=2 THEN IF (N-N3)*(LASTN-N3)<=0E0 THEN DO;
/* THE ERRORS HAVE NOW DECREASED TO N3 WHICH CORRESPONDS
   TO THE THIRD TRANSITION IN THE ERROR EQUATION.      */
  T3=T;
  NCT3=NC;
  STAGE=3;
  END;
LASTN=N;
END LOOP;
CALL PRINTGRAPH;
PUT SKIP DATA(T1,T2,T3);
END CASE3;
```

```
(SIZE):
 PRINTGRAPH:PROC;
 DECLARE I ,  REDUCTION_FACTOR FLOAT;
 DECLARE BIG;
 DECLARE PRINTLINE CHAR(80);
 DECLARE (FLOOR,SUBSTR) BUILTIN;
 /* FIND MAX OF ARRAY NT_TAB                                    */
 BIG=0;
 DO I=0 TO FLOOR(FIN/DGRAPHT)  ;
 IF NT_TAB(I)>BIG THEN BIG=NT_TAB(I);
 END;
 ON SIZE SYSTEM;
 ON SUBSCRIPTRANGE SNAP BEGIN; PUT DATA; STOP; END;
 ON STRINGRANGE SNAP BEGIN; PUT DATA; STOP; END;
 REDUCTION_FACTOR =50/BIG;
 (SIZE):  PUT EDIT((I/REDUCTION_FACTOR
 DO I= 0 TO 100 BY 10          ))(COL(14),F(7,2), 10F(10,2));
 PUT EDIT  ((6)'¬'||'|', ((9)'¬'||'|' DO I = 1 TO 10))
           (COL(14),A,10A(10));
 PRINTLINE=' ';
 SUBSTR(PRINTLINE,REDUCTION_FACTOR*N_TAB(0)+1,1)='N';
 SUBSTR(PRINTLINE,REDUCTION_FACTOR*NC_TAB(0)+1,1)='C';
 SUBSTR(PRINTLINE,REDUCTION_FACTOR*NT_TAB(0)+1,1)='T';
 PUT  SKIP(0) EDIT(PRINTLINE)(COL(20),A);
 DO I=1 TO COUNT;
 PRINTLINE=' ';
 SUBSTR(PRINTLINE,1,1)='|';
 SUBSTR(PRINTLINE,REDUCTION_FACTOR*N_TAB(I)+1,1)='N';
 SUBSTR(PRINTLINE,REDUCTION_FACTOR*NC_TAB(I)+1,1)='C';
 SUBSTR(PRINTLINE,REDUCTION_FACTOR*NT_TAB(I)+1,1)='T';
 IF MOD(I,10)=0 THEN PUT EDIT(I*DGRAPHT,PRINTLINE)
                    (COL(9),F(10,2),X(1),A);
 ELSE
 PUT EDIT(PRINTLINE)(COL(20),A);
 END;
 PUT PAGE;
 PUT EDIT ('N','NC','NT')
  (COL(10),3A(10))
 ((N_TAB(KK),NC_TAB(KK),NT_TAB(KK) DO KK=0 TO COUNT))
 (COL(8),3F(10,2))       ;
 RETURN;
 END PRINTGRAPH;
 END MODELS ;
```

CASE 1

```
NO = 100              NP = 125            N1 = 25
A1 = 0.125            A1P = 0.0350        K1 = 10
K1P = 12.5            K2 = 0.150          DT = 0.100
DGRAPHT = 1           FIN = 50
```

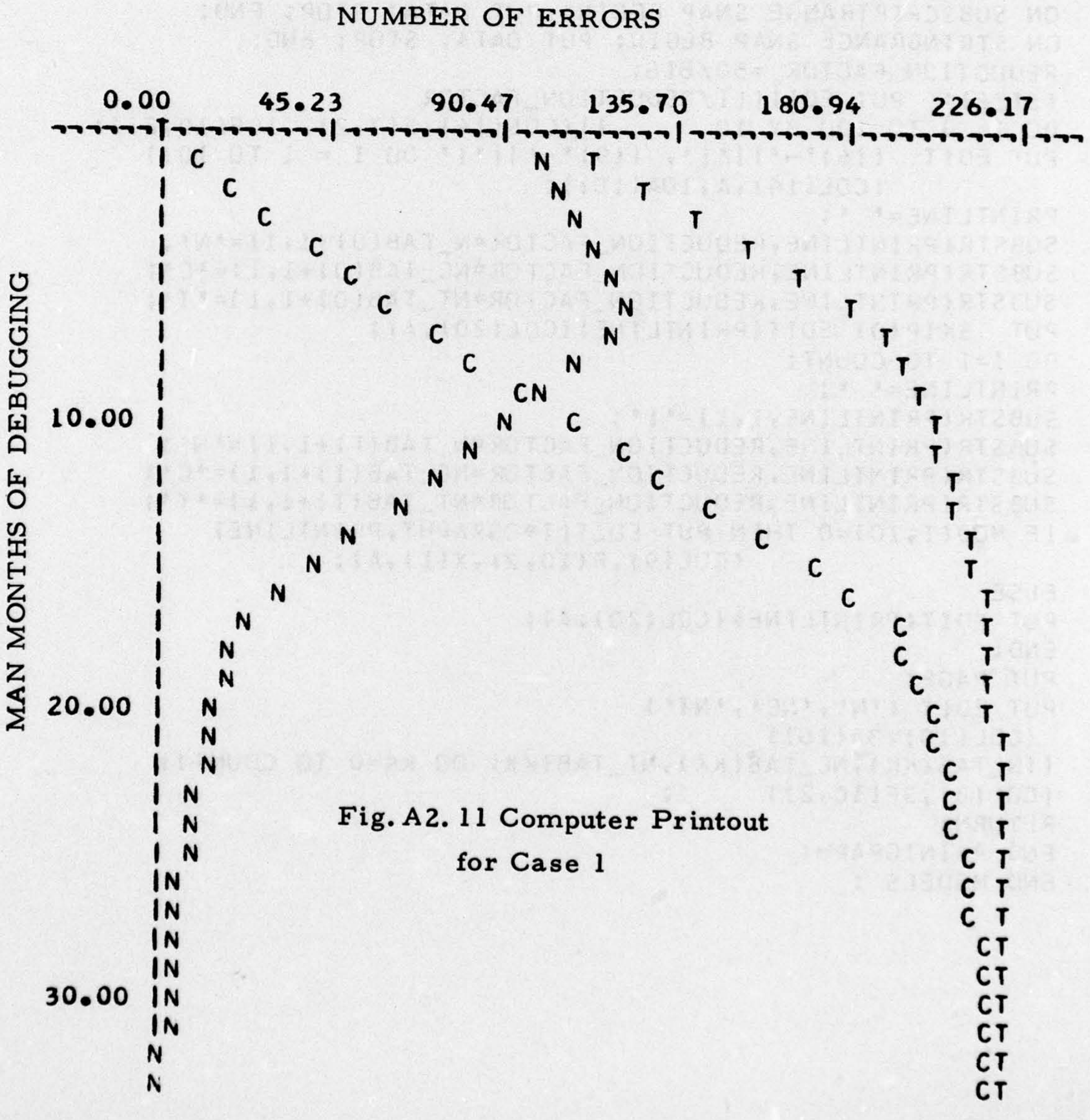


Fig. A2. 11 Computer Printout for Case 1

```
                         CASE 2

NO = 100                 N1 = 25                  A1 = 0.0250
K1 = 10                  K2 = 0.150               DT = 0.100
FIN = 50                 DGRAPHT = 1
```

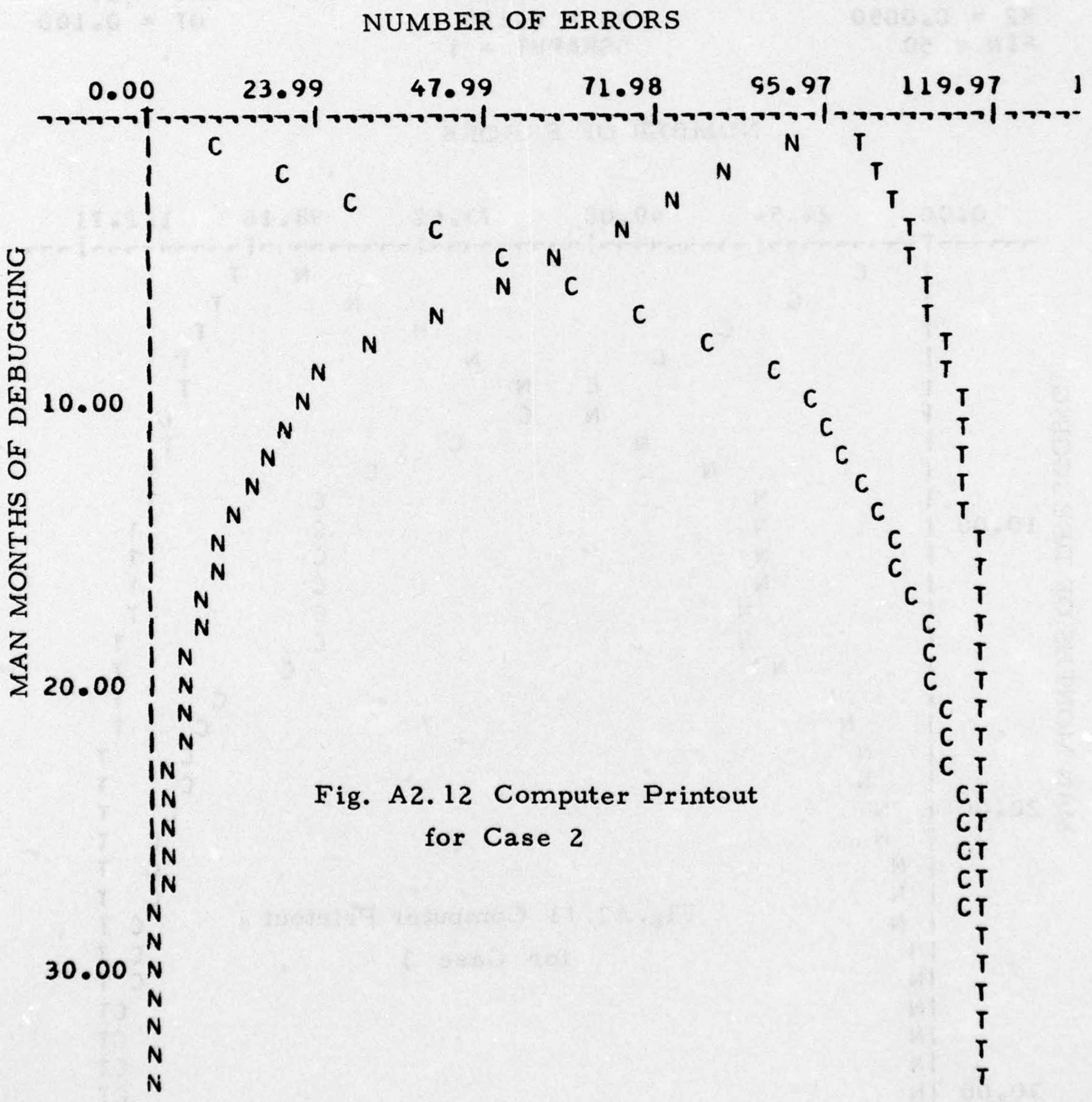


Fig. A2.12 Computer Printout for Case 2

CASE 3

```
NO = 100                  N1 = 25                N2 = 28
N3 = 15                   A1 = 0.0250            K1 = 10
K2 = 0.0050               K3 = 0.150             DT = 0.100
FIN = 50                  DGRAPHT = 1
```

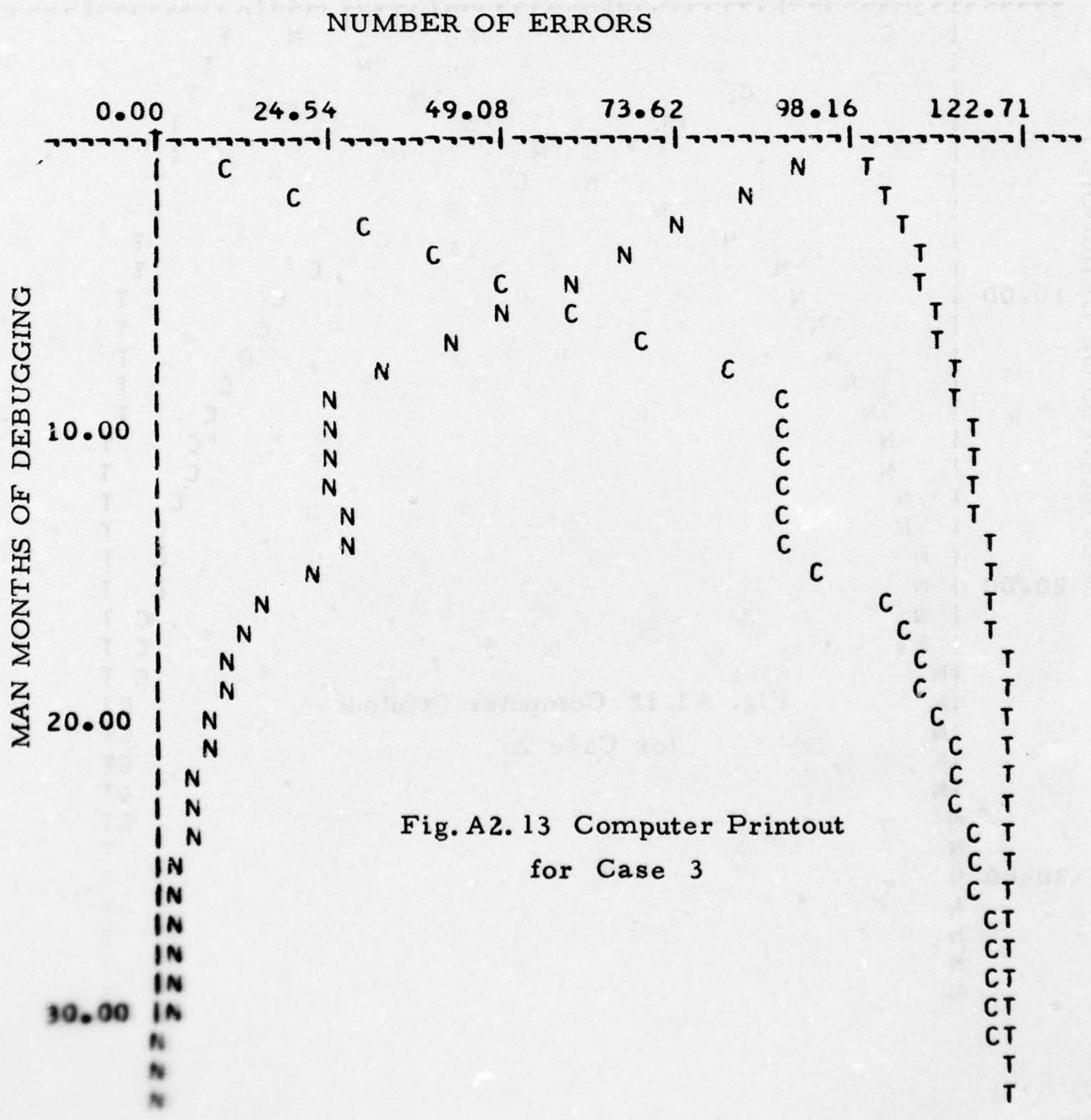


Fig. A2. 13 Computer Printout for Case 3

REFERENCES

[.1] Martin L. Shooman, "Probabilistic Models for Software Reliability Prediction, Conference on Statistical Methods for the Evaluation of Computer System Performance, " Brown University, Nov. 1971, Frieberger edition, Academic Press, N. Y. C.

[.2] Z. Jelinski and P. B. Moranda, "Software Reliability Research, " published in Statistical Computer Performance Evaluation, 1972, Frieberger edition, Academic Press, New York.

[3] Martin L. Shooman, "Operational Testing and Software Reliability Estimation During Program Development, " 1973, IEEE Symposium on Computer Software Reliability, New York City, April 30-May 2, 1973.

[4] Martin L. Shooman, et al, unpublished memoranda on Error Generation Models, Bell Laboratories, March 1973.

[5] John D. Musa, "A Theory of Software Reliability and Its Applications, " IEEE Transactions on Software Engineering, Vol. SE-1, No. 3, Sept. 1975.

[6] M. L. Shooman and M. I. Bolsky, "Types, Distributions and Test and Correction Times for Programming Errors, " Proceedings, 1975 International Conference on Reliable Software, Los Angeles, April 21-23, 1975.

[7] J. Dickson, J. Hesse, A. Kientz, and M. Shooman, "Quantitative Analysis of Software Reliability, " 1972 Annual Reliability Symposium Proceedings, IEEE, January 1972.

[8] F. Akiyama, "An Example of Software System Debugging, " IFIP Congress 1971, Ljubljana, Yugoslavia, August 1971.

[9] Frederick P. Brooks, Jr., "How does the project get to be a year late? -- One day at a time, " Datamation, December 1974.

[10] Daniel Allen Rankin, "A Model of the Cost of Software Development for the Apollo Spacecraft Computer, " Submitted in partial fulfillment of the requirements for the degree of Master of Science at the "Massachusetts Institute of Technology, " June 1972.

[11] Martin L. Shooman, "Probabilistic Reliability: An Engineering Approach, " McGraw-Hill, 1968.

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	*metre per second squared*	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	*candela per square metre*	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	*joule per kilogram-kelvin*	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

☆U.S. GOVERNMENT PRINTING OFFICE: 1977-714-025/100